HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
Fakultät IV
Wirtschaft und
Informatik

Entwicklung einer natürlichsprachlichen, logischen Programmierumgebung für den Einsatz im Schulunterricht

Masterarbeit
zur Erlangung des akademischen Grades
Master of Science (M. Sc.)
im Studiengang Angewandte Informatik

Verfasser:	Marcel Kaufmann
Matrikelnummer:	1345380
Erstprüfer:	Prof. Robert Garmann
Zweitprüfer:	Prof. Elisabeth Dennert-Möller
Hochschule:	Hochschule Hannover
Datum:	12. Februar 2018

Danksagung

Ein großer Dank geht an die Grundschule Poggenhagen und an die Schulleiterin Birgit Bösche, die es mir ermöglichte im Rahmen des Schulunterrichts die Forschung dieser Arbeit durchzuführen. Des Weiteren möchte ich meinen beiden Prüfern Prof. Dr. Elisabeth Dennert-Möller sowie Prof. Dr. Robert Garmann herzlich danken, die vor allem in der Erarbeitung des Themas mit ihren kritischen Nachfragen und Anmerkungen die Selbstreflexion begünstigten sowie mit Einwänden und Ideen zur Weiterentwicklung beigetragen haben. Ein großer Dank geht außerdem an Sonja Teichmann, die als Lehrkraft in den Unterrichtsstunden unterstützte und für ein Interview zur Verfügung stand. Abschließend bedanke ich mich bei allen Korrekturlesenden für ihre Zeit und Anmerkungen.

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Konzeption und Umsetzung einer Entwicklungsumgebung als Android-App für den Einsatz im Schulunterricht. Die Anwendung ist mit der natürlichen Sprache zu bedienen und stellt eine Entwicklungsumgebung für das logische Programmierparadigma zur Verfügung. Die entstandene Software wird in einem Feldversuch mit Kindern der 4. Klassenstufe an einer Grundschule erprobt. Da Änderungen und neue Erkenntnisse möglichst effizient in die Software einfließen sollen, wurde die App nach der Clean Architecture umgesetzt. Die Ergebnisse aus der Feldforschung zeigen, dass die Schülerinnen und Schüler mit hoher Motivation an einem fachlichen Thema gearbeitet haben. Sie erlernten die Konzepte der Fakten, Regeln und der Wissensbasis und vertieften damit ihr Wissen zu geometrischen Formen in der Mathematik. Der Architekturansatz der App stellt dessen Vorteile deutlich an Fallbeispielen dar. Die Arbeit zeigt das spannende Feld der angewandten Wissenschaft, das die beiden Disziplinen Informatik und Didaktik verknüpft.

Inhaltsverzeichnis

1. Einleitung

> „We think it is time to take another look at an old dream -- that one could program a computer by speaking to it in natural language.“
>
> (Lieberman und Liu (2006))

Digitale Assistenten wie Siri, Google Assistent oder Alexa sind fester Bestandteil auf den Smartphones und Smarthome-Geräten. Diese werden unter anderem mit der natürlichen Sprache bedient. Somit wird das Sprechen mit den diversen Hardwaregeräten bei steigender Verbreitung der Smarthome-Geräte signifikant wachsen. Nicht nur interessierte Erwachsene, sondern auch im Besonderen Kinder, die mit diesen Geräten aufwachsen, könnten es in Zukunft als selbstverständlich ansehen, mit Geräten natürlichsprachlich zu kommunizieren. Daher wird dies ein Thema, das immer stärker in Schulen und Bildungseinrichtungen aufgegriffen werden sollte, um die Mediennutzung in dieser Thematik anzusprechen.
Berührungspunkte zur Programmierung können Kinder in vielen verschiedenen Situationen erleben. Ein standardisiertes Vorgehen, vor allem für Schülerinnen und Schüler (SuS) der Primarstufe, ist allerdings noch nicht etabliert. Bildungsstandards im Primarbereich für das Fach Informatik wurden jedoch im September 2017 verabschiedet und versprechen eine Standardisierung (Gesellschaft für Informatik (GI) e. V., 2017). Bisheriger Fokus hinsichtlich der Programmierung ist das algorithmische, prozedurale bzw. imperative Paradigma. Ein deklarativer Programmieransatz mit der natürlichsprachlichen Programmierung wurde bisher noch nicht zusammen in einen Schulkontext gesetzt. Bei der Konzeption eines neuen Werkzeugs für den Einsatz im Schulunterricht müssen verschiedene Randbedingungen beachtet werden. Neben dem Ziel einer gesteigerten Medienkompetenz und dem Erlernen von Programmiergrundlagen ist die Integration in den bestehenden Strukturen wichtig. Ist es möglich, eine Programmiersoftware (Entwicklungsumgebung) als Werkzeug im Schulalltag zu integrieren? Für die Schülerinnen und Schüler soll nicht so sehr das Produktwissen, wie ein Werkzeug zu bedienen ist, sondern vielmehr das Konzeptwissen im Vordergrund stehen. Modrow und Strecker (2016)

sehen hier eine besondere Chance für die Informatik im Schulalltag und die Mächtigkeit der Werkzeuge, die für die Vermittlung besonders im Vordergrund stehen. „Vor allem aber ändert sich mit der Mächtigkeit der Werkzeuge auch das Verhältnis zwischen dem Aufwand, der für Werkzeugkenntnisse, etwas das Erlernen der Syntax einer Programmiersprache, erforderlich ist, und deren Anwendung. Wenn Informatiksysteme zunehmend in der Lage sind, menschengeeignete Sprache zu verstehen, dann können auch immer mehr Menschen Computer als Werkzeuge benutzen. Der Weg zum Pflichtfach für alle wird geebnet“ (Modrow und Strecker, 2016, S. 25). Weiter beschreiben sie den Vorteil für die Lernenden, die durch ihre Rolle zu Konstrukteuren werden, die miteinander für andere Probleme analysieren und lösen. Zusätzlich werden auch konkrete Resultate erzielt und Produkte erzeugt. Die Ergebnisse der Überlegungen und Auswirkungen können getestet, erprobt und zum Teil verallgemeinert werden.
Eine solche Verallgemeinerung ist vor allem durch das Erkennen und Formulieren von Regeln möglich (siehe Abschnitt 2.2.4 und 2.2.5). Das Erlernen von Problemlösetechniken ist in diesem Kontext besonders hervorzuheben. Dabei „[...] eignet sich [das prädikative programmiersprachliche Denkschema] besonders gut zur Einführung in das Problemlösen mit Methoden und Mitteln der Informatik, da das Aufstellen von Regeln für den Problemlöseprozess hier explizit erlernt wird“ (Schubert und Schwill, 2011, S. 91).
Diese Arbeit zeigt an einer neu konzipierten und entwickelten Entwicklungsumgebung, wie ein solch *mächtiges Werkzeug* aussehen kann. Dieses wird im Kontext des Schulunterrichts mit Kindern der 4. Klassenstufe in einem Feldversuch untersucht und bewertet. Da der gewählte natürlichsprachliche Ansatz im deklarativen Bereich bislang weitgehend unerforscht und unbeachtet ist, entsteht die Entwicklungsumgebung mit einer Softwarearchitektur, die schnelle Änderungen und Anpassungen ermöglichen soll. So könnten neue Erkenntnisse aus dieser oder anderen Arbeiten in die Anwendung einfließen.
Durch den interdisziplinären Ansatz der technischen Informatikaspekte und der didaktischen Überlegungen bezüglich Einsatz und Integration in den Schulalltag zeigt diese Arbeit eine Verknüpfung der Disziplinen.

2. Technische Grundlagen

Das folgende Kapitel zu technischen Grundlagen ist eines von zwei Grundlagenkapiteln. Da diese Arbeit sowohl einen hohen Anspruch an den Informatikteil stellt, als auch durch die Forschungsfragen didaktische Aspekte beinhaltet, bilden die beiden Grundlagenkapitel eine Basis zum Verständnis der Teilgebiete. Im Folgenden werden die technischen, informatischen Aspekte zu Android, Kotlin, Programmierparadigmen, Softwaredesignmethoden und Architekturkonzepten angesprochen sowie erklärt. Des Weiteren gibt dieses Kapitel einen Einblick in Prolog sowie Expertensysteme und beschreibt abschließend, wozu *natural language processing* (NLP) verwendet wird. Um die Qualität der technischen Aspekte bewerten zu können, wird darüber hinaus an dieser Stelle die Qualitätsnorm zu Software(-Produkten) vorgestellt.

2.1. Android

Die Programmierumgebung soll als Android-App für Tablets und Smartphones umgesetzt werden. Um technische Ziele und auf Android bezogene Architekturentscheidungen zu verstehen, sollte Grundlagenwissen zu android-spezifischen Mechanismen vorhanden sein. Android ist ein mobil-orientiertes, auf Linux bzw. Unix basierendes Betriebssystem (vgl. Louis und Müller, 2016, S. 177). Eine Java-Virtual-Maschine (JVM) abstrahiert die Ebene der Hardware. Durch diese Abstraktion lassen sich mit Hilfe von Java, XML-basierten Layouts und Konfigurationsdateien Anwendungen für dieses Betriebssystem entwickeln (vgl. Ostrander, 2012, S. 3). Spätestens seit der Entwicklerkonferenz Google I/O 2017, auf der Kotlin (vgl. Abschnitt 2.2.1) als offiziell unterstützte Sprache aufgenommen wurde, ist das Programmieren neben Java auch in dieser JVM-Sprache möglich beziehungsweise sinnvoll.

2.1.1. Intent

Ein Intent ist ein Benachrichtigungsobjekt, das für die Anfrage an eine andere App-Komponente verwendet wird. Darüber hinaus erleichtert es die Kommunikation zwischen diesen Komponenten (vgl. Android-Developer, 2017a). Als solche kann in diesem Fall eine andere App oder beispielsweise eine zweite Activity innerhalb der gleichen App angesehen werden. Es existieren zwei Arten von Intents.
Der explizite Intent wird über den voll qualifizierten Klassennamen angegeben (Beispiel Activity: de.hsh.app.MainActivity siehe Quelltext 2.1). Ein typischer Anwendungsfall ist das Starten einer anderen Activity in der eigenen App (vgl. ebd.).

Quelltext 2.1.: Expliziter Intent - Aufruf der IDEActivity

```
val intent = Intent(mainActivity, IDEActivity::class.java)
mainActivity.startActivity(intent)
```

Beim impliziten Intent wird nicht die konkrete Komponente, sondern eine Aktion angefragt. Auf diesen Broadcast in das System geben alle Apps (bzw. dessen Services und Activities), die diese angefragte Aktion bedienen können, Rückmeldung. Quelltext 2.2 zeigt einen Ausschnitt aus der Manifest-Datei einer App. In diesem werden alle Konfigurationen zur App hinterlegt. Dort spezifiziert sind unter anderem die enthaltenen Activities und wie diese aufgerufen werden können. Der Intent-Filter spezifiziert, dass die Activity *ShareActivity* auf Aktionen *SEND* mit dem Typ *text/plain* reagieren soll.

Quelltext 2.2.: Ausschnitt aus dem Manifest - Impliziter Intent aus Android-Developer (ebd.)

```
<activity android:name="ShareActivity">
 <intent-filter>
  <action android:name="android.intent.action.SEND"/>
  <category android:name="android.intent.category.DEFAULT"/>
  <data android:mimeType="text/plain"/>
 </intent-filter>
</activity>
```

Sollten mehrere Apps existieren, die diesen Intent-Filter spezifiziert haben, kann der Nutzer auswählen, welche App darauf reagieren soll. Ein Beispiel ist das Versenden von Textnachrichten. Eine andere App ruft den impliziten Intent wie in Quelltext 2.3 gezeigt auf.

Quelltext 2.3.: Impliziter Intent - Aufruf der *SEND*-Aktion aus Android-Developer (ebd.)

```
val sendIntent = Intent()
sendIntent.action = Intent.ACTION_SEND
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage)
sendIntent.type = "text/plain"
if (sendIntent.resolveActivity(mainActivity.packageManager)
 ↪  != null) {
 mainActivity.startActivity(sendIntent)
}
```

Bundle-Daten

Neben dem reinen Aufruf der Activities können in die Benachrichtigungsobjekte (Intents) zusätzliche Informationen integriert und somit an die nächste Activity übergeben werden. Diese sind in das *Extra* Feld des Intents zu schreiben (vgl. Louis und Müller, 2016, S. 251f). Es ist entweder möglich die Daten als Schlüssel-Wert-Paare hinzuzufügen oder in einem *Bundle*-Objekt zu hinterlegen. Dieses Bundle-Objekt nimmt neben den primitiven Datentypen (String, char, long, double, int, usw.) auch Objekte entgegen, die das *Parcelable*-Interface implementieren. Dies ist eine android-spezifische Form der Serialisierung/Deserialisierung. Serialisierung ist das Umwandeln eines Objektes in Zeichenketten. Deserialisierung bezeichnet wiederum das Zurückübersetzen der Zeichenkette in das Objekt. Anwendung findet dies beim Persistieren oder Versenden von Objekten. Quelltext 2.4 zeigt einen expliziten Aufruf einer neuen Activity mit einem Intent, der mit Informationen durch ein Bundle-Objekt erweitert wurde.

Quelltext 2.4.: Ein Intent erhält zusätzliche Informationen über ein Bundle

```
val intent = Intent(mainActivity, IDEActivity::class.java)
val b = Bundle()
b.putParcelable(Constants.MINI_WORLD_PROFILE_BUNDLE_KEY,
 ↪  miniWorldProfileViewModel)
intent.putExtras(b)
mainActivity.startActivity(intent)
```

2.1.2. App-Lebenszyklus

Sobald eine Android-App Inhalte visuell darstellt, geschieht dies in einer View. Die View besteht im Android Kontext aus einer Activity oder einem Fragment mit entsprechendem Layout. Das Layout wird entweder programmiert (in Java/Kotlin), spezifiziert (in XML) oder in einer Mischung dieser beiden integriert. Eine *Activity* im Sinne des Android-Frameworks ist eine einzelne, fokussierte Intention, die ein Nutzer verfolgt und durchführen kann. Nahezu alle Aktivitäten interagieren mit dem Nutzer. Somit erstellt die Activity-Klasse den Bildschirmbereich, in dem die Nutzerinteraktion stattfindet (vgl. Android-Developer, 2017b).
Activities durchlaufen einen Lebenszyklus, der aus verschiedenen Status besteht. Deren Benennung und eine Auswahl der hier relevanten Übergänge sind in Abbildung 2.1 dargestellt. Beim Eintritt in den jeweiligen Status ruft das Android System die entsprechende Methode in der Activity auf.

Sobald eine Activity (A) initialisiert wird, ruft das Android System die *onCreate()*-Methode auf. Diese Methode wird zum Initialisieren der Layouts, Menüs oder anderer Objekte benötigt, die während des Lebenszyklus vorhanden sein sollen. Nachfolgend wechselt der Status in *onStart()*. Sobald die Layoutkomponenten den Fokus auf dem Bildschirm erhalten, ruft das Android System *onResume()* auf. Die Anwendung ist nun im Vordergrund und bereit für Nutzerinteraktionen. Sobald eine andere Activity (B) (der selben App, einer anderen App oder das System durch Navigation auf den Home-Screen des Smartphones) in den Vordergrund gerufen wird, führt die Activity (A) *onPause()* und *onStop()* aus. Activity (A) ist nun nicht mehr sichtbar, aber im Speicher vorhanden und bereit erneut angezeigt zu werden. Jedoch erhält sie vom System das *KillableAfter*-Flag (vgl. Louis und Müller, 2016, S. 182). Dies bedeutet, falls diese Activity Platz im Speicher belegt, den eine andere App (Activity) im Vordergrund benötigt, dann beendet das Android System Activity (A), ohne dass die *onDestroy()*-Methode aufgerufen wird. Ansonsten wird sie durch den Aufruf von *finish()* (durch sich selbst oder einer anderen Activity) über *onDestroy()* beendet. Sollte die bestehende Instanz aus dem Hintergrund wieder in den Vordergrund gerufen werden, so ruft das Android System *onRestart()* und nachfolgend *onStart()* auf. Sobald die Activity (A) wieder sichtbar wird, folgt der *onResume()*-Aufruf und die Activity (A) ist bereit für die Interaktion des Nutzers. Zu beachten ist, dass bei einem *orientation change* (dem Wechsel von Portrait auf Landscape Ansicht oder umgekehrt) die Statusübergänge bis inklusive *onDestroy()* und nachfolgend direkt die *onCreate()*-Methode aufgerufen werden. Objekte, die wichtig für den aktuellen Status sind, müssen gespeichert werden.

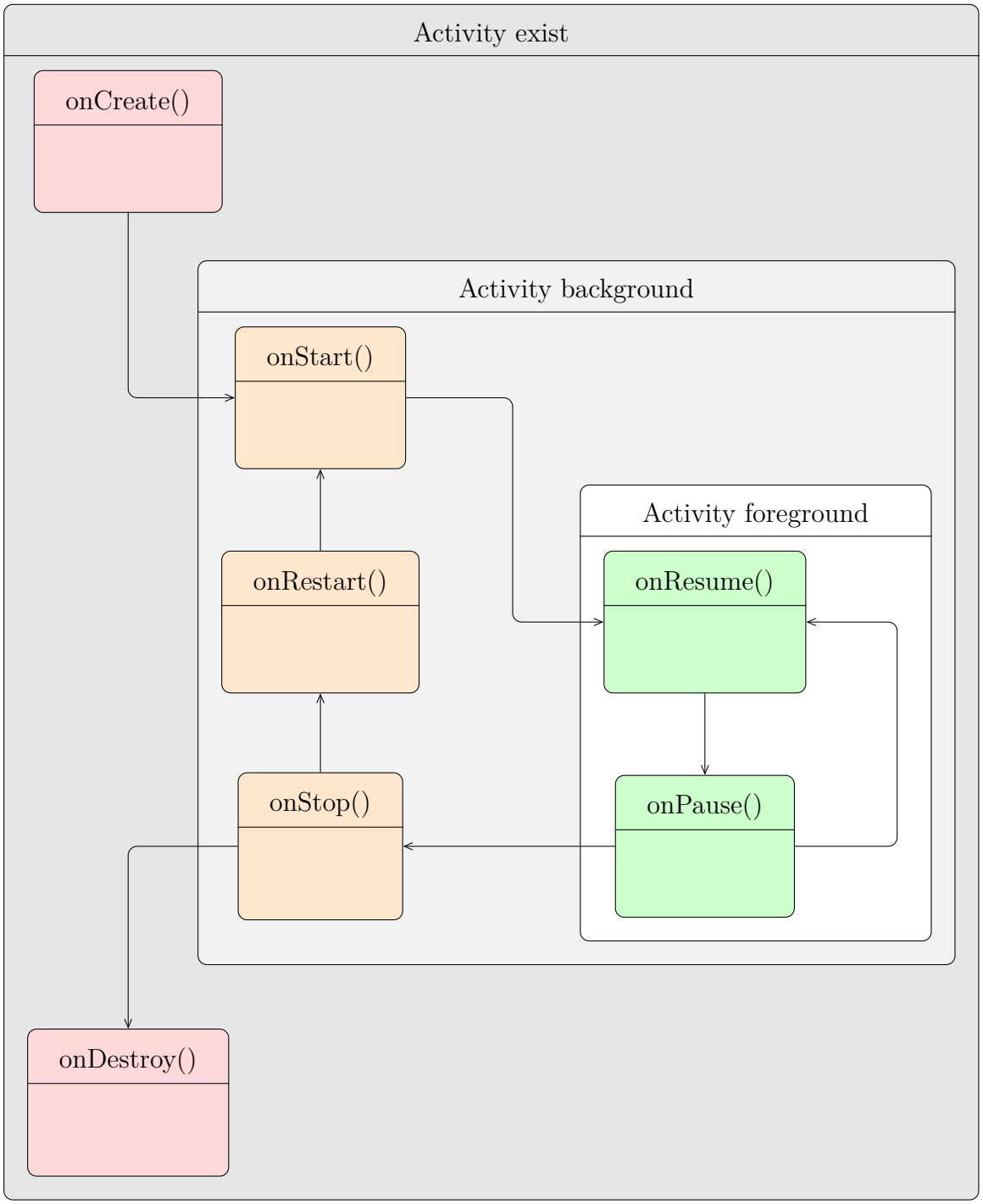


Abbildung 2.1.: Vereinfachter Activity Lebenszyklus nach Louis und Müller (2016, S. 181)

2.2. Programmiersprachen und Paradigmen

Dieser Abschnitt stellt die beiden für die Arbeit wichtigen Programmiersprachen Kotlin und Prolog vor. Um Programmbeispielen folgen zu können, ist es außerdem nötig, Grundlagen des funktionalen Programmierparadigmas und den Einsatzzweck von Reactive Programming zu verstehen. Da in Folge der Arbeit eine logische Programmierumgebung entstehen soll, wird Prolog als Vertreter für dieses Paradigma eingeführt.

2.2.1. Kotlin

Die gewählte Entwicklungssprache ist Kotlin. „Kotlin is a JVM based language developed by JetBrains[1], [...] "(Leiva, 2016, S. 5) und erhält seit der Google I/O 2017 den offiziellen Einzug in die Android-Programmierung. Die Projektquelle[2] sowie Quellen wie Leiva (ebd., S. 5) arbeiten unter anderem folgende Besonderheiten heraus.

- Kotlin sei *expressiver*. Der Entwickler könne mit weniger Code auskommen, um die gleiche Funktionalität zu implementieren. Boilerplate Code wird somit reduziert.
- Kotlin sei sicherer während der Laufzeit, da durch Checks des Compilers Nullpointer-Exceptions verhindert werden. Eine Nullpointer-Exception ist ein Zugriff auf eine Objektreferenz, die nicht instanziiert und somit nicht im Speicher ist. Objekte können Null sein, müssen aber als solche deklariert werden. Bei Nutzung des Objektes muss der Entwickler die Instanziierung prüfen.
- Kotlin ist eine funktionale Programmiersprache. Wie beispielsweise Scala ist sie nicht *pure*, aber implementiert viele funktionale Aspekte, die den Code fehlerresistenter und leichter lesbar machen sollen. Dieses Argument wird in Abschnitt 2.2.2 separat behandelt.
- Kotlin ist interoperabel zu JVM-Sprachen. Dies bedeutet, dass beispielsweise bestehende Java-Bibliotheken durch Kotlin aufgerufen und genutzt werden können. Es ist möglich, ein Projekt sowohl mit Java- als auch mit Kotlin-Quellcode koexistent zu verwenden.

Diese Eigenschaften gelten sowohl für den allgemeinen Vergleich zur Java-Entwicklung, aber im Besonderen hinsichtlich Android, da diese Plattform auf dem Java 6 Bytecode aufsetzt. Kotlinquellcode kompiliert ebenfalls zu dieser Bytecodeversion. Damit ist eine

[1] https://www.jetbrains.com/
[2] https://kotlinlang.org

vollständige Abwärtskompatibilität gewährleistet. Durch die aufgezählten Besonderheiten von Kotlin, die allgemeinen Vorteile funktionaler Programmierung (siehe Abschnitt 2.2.2) und Googles Entscheidung, Kotlin als offiziell unterstützte Programmiersprache aufzunehmen, wird diese Sprache zu einer sinnvollen Alternative zur Entwicklung in Java.

2.2.2. Funktionales Programmierparadigma

Kotlin ist eine funktional orientierte Programmiersprache (vgl. 2.2.1). Die im Rahmen dieser Arbeit umgesetzte Anwendung folgt somit diesem Ansatz.
Funktionale Programmierung gehört neben der imperativen und logischen Programmierung zu den drei fundamentalen Programmierparadigmen. Sie ist eng verbunden mit dem Funktionsbegriff der Mathematik (vgl. Backfield, 2014, S. viii) und zeichnet sich durch das Umformen von Funktionen, nicht aber durch das Verändern verschiedener Status aus (vgl. Gabbrielli und Martini, 2010, S. 333f). Darüber hinaus können nach Backfield (2014, S. 1) sieben allgemeine Konzepte identifiziert werden.

1. First-class functions
2. Pure functions
3. Recursion
4. Immutable variables
5. Nonstrict evaluation
6. Statements
7. Pattern Matching

Backfield sieht einen großen Vorteil im leichteren Verständnis des Quellcodes. Funktionales Programmieren zwinge den Entwickler dazu, vor und während des Entwickelns über die Abstraktion nachzudenken und diese anzupassen (vgl. ebd., S. ix). Die Anforderungen bzw. Erwartungen an Software steigen und der mathematische Ansatz zu verlässlichen Algorithmen könne dem Entwickler die Möglichkeit geben, dem gerecht zu werden. Darüber hinaus sei es möglich, die mathematischen Konzepte zu nutzen, um potentielle Schwächen aufzudecken (vgl. ebd., S. x). Die Wiederverwendbarkeit und das Testen auf einer höheren Abstraktionsebene sind für ihn die Hauptvorteile funktionaler Programmierung.

Ford (2014, S. 11ff) zeigt anschaulich an einem Beispiel die unterschiedlichen Konzepte von imperativer und funktionaler Programmierung. Die Aufgabe ist folgende. Gegeben

sei eine Liste von Namen. Einige Einträge enthalten nur ein einziges Zeichen. Ziel ist das Zurückgeben einer komma-separierten Zeichenkette, in der die Einträge mit einzelnen Zeichen verworfen und die verbliebenen Namen mit einem Großbuchstaben beginnend ausgegeben werden. Nach imperativer Vorgehensweise ist die Liste in einer Schleife zu verarbeiten. Somit führt der Entwickler mehrere Operationen im Schleifenbauch nacheinander durch (vgl. Quelltext 2.5). Die Liste wird gefiltert, transformiert und konvertiert.

Quelltext 2.5.: Verarbeiten einer Liste in Java - Imperatives Paradigma

```
static List<String> employees = Arrays.asList("neal", "s",
 ↪ "stu", "j", "rich", "bob", "aiden", "j", "ethan", "liam",
 ↪ "mason", "noah", "lucas", "jacob", "jayden", "jack");

private static String cleanNames(List<String> listOfNames) {
  StringBuilder result = new StringBuilder();
    for (int i = 0; i < listOfNames.size(); i++) {
      if (listOfNames.get(i).length() > 1) {
        result.append(capitalizeString(listOfNames.get(i)));
        result.append(",");
      }
    }
  return result.substring(0, result.length() - 1);
}

private static String capitalizeString(String s) {
  return s.substring(0, 1).toUpperCase() + s.substring(1,
   ↪ s.length());
}
```

Eine funktionale Herangehensweise kategorisiert die Probleme als Operationen auf der Liste auf niedrigerer Ebene. Es obliegt dem Entwickler, diese speziell benötigte Ausprägung zu programmieren. Durch die Nutzung *Higher-Order-Functions* wie *filter* implementiert der Entwickler das gewünschte Verhalten der Filter-Operation. Die beschriebene Aufgabe ist in Kotlin wie folgt gelöst (siehe Quelltext 2.6).

Quelltext 2.6.: Verarbeiten einer Liste in Kotlin - Funktionales Paradigma

```
fun main(args: Array<String>) {
  val employees = listOf("neal", "s", "stu", "j", "rich",
          "bob", "aiden", "j", "ethan", "liam", "mason",
          "noah","lucas", "jacob", "jayden", "jack")
  val result: String = employees
  .filter { it.length > 1 }
  .joinToString(",") { it.capitalize() }
}
```

Da die Funktion *joinToString* ein Kotlin Sprachfeature ist, zeigt Quelltext 2.7 die Umsetzung in den funktional typischen Operationen *filter*, *map* und *reduce* (vgl. Ford, 2014, S. 31ff).

Quelltext 2.7.: Funktionale Umsetzung mit *filter*, *map* und *reduce*

```
fun main(args: Array<String>) {
 val employees = listOf("neal", "s", "stu", "j", "rich",
  ↪ "bob", "aiden", "j", "ethan", "liam", "mason", "noah",
  ↪ "lucas", "jacob", "jayden", "jack")
 val result: String = employees
  .filter { it.length > 1 }
  .map { it.capitalize() }
  .reduce { all, next -> all + "," + next }
       }
```

Auf die Konzepte *First-class functions*, *Pure functions* und *Immutable variables* wird im Folgenden etwas genauer eingegangen.

Funktionen als Objekte - First-class functions

> „First-class functions are functions treated as objects themselves, meaning we can pass a function as a parameter to another function, returning a function from a function, or store a function in a variable“ (Backfield, 2014, S. 5).

Um Funktionen anderen Funktionen als Parameter zu übergeben, werden *Higher-Order-Functions* benötigt. Dies ist nur unter der Annahme möglich, dass Funktionen Werte

sind. Wie alle Werte, ähnlich *Integers*, *Strings*, *Listen* [und Objekte], können sie einer Variablen zugewiesen werden. Damit ist die Funktion Teil der Datenstruktur und ermöglicht dem Entwickler, diese als Parameter zu übergeben (vgl. Chiusano, 2014, S. 19).

Reinheit - Pure functions

Funktionen werden für einen bestimmten Zweck geschrieben. Jede Funktion ist für eine Aufgabe konzipiert, die sie für einen bestimmten Input durchführt. Sobald Variablen außerhalb des Funktionsscopes verändert werden, nennt sich dieses Phänomen *Seiteneffekt*. Reine Funktionen hingegen liefern bei gleichem Input immer denselben Output (vgl. Backfield, 2014, S. 25). Dies kann nur gewährleistet werden, solange die Funktion auf keine externen Quellen zugreift oder anderweitig Veränderungen hervorruft. Ausgenommen ist hier der Rückgabewert der Funktion. Chiusano definiert dies wie folgt.

> „A function f with input type A and output type B [...] is a computation that relates every value a of type A to exactly one value b of type B such that b is determined solely by the value of a. Any changing state of an internal or external process is irrelevant to computing the result *f(a)*“.
>
> ((Chiusano, 2014, S. 9))

Unveränderbarkeit - Immutable variables

Laut Ghosh (vgl. 2017, S. 7) und Ford (vgl. 2014, S. 137) soll in der funktionalen Programmierung so viel unveränderbar entwickelt werden wie möglich.
Wenn über Variablen gesprochen wird, dann sind meist veränderbare Variablen gemeint. Sie sind, wie der Name besagt, variabel. Das bedeutet, dass es möglich ist, verschiedene Werte in einer Variablen zu speichern und diese wiederzuverwenden (vgl. Backfield, 2014, S.43f). Quelltext 2.8 zeigt eine Variable *text*, die zwei verschiedene Werte zugewiesen bekommt.

Quelltext 2.8.: Veränderbare Variable

```
var text = "Hallo"
text = "Welt"
```

Die verschiedenen Programmiersprachen gehen mit der Unveränderbarkeit unterschiedlich um. In Java kann über das Schlüsselwort *final* eine Klasse oder Parameter unveränderbar deklariert werden. Somit ist es nicht möglich, eine einmal instanziierte Variable

erneut zuzuweisen. Von einer finalen Klasse kann keine Unterklasse abgeleitet werden, um so Methoden oder Parameter zu überschreiben. In Kotlin dagegen sind Klassen standardmäßig final. Parameter werden durch *var* (veränderbar) und *val* (unveränderbar) deklariert.
Oft wird davon ausgegangen, dass es die Komplexität erhöht, wenn der Quellcode auf unveränderbaren Variablen basiert. Jedoch helfe es, der Komplexität entgegenzuwirken. Vor allem könne die Anfälligkeit von Fehlern minimiert werden, da nur wenige Seiteneffekte vorhanden sind (vgl. ebd., S. 54). Sollte auf (fast) alle veränderbaren Variablen verzichtet werden, so gäbe es vor allem in Anwendungen, die mehrere Prozesse verwenden, weniger bis keine typischen Probleme (*race conditions*, *deadlock conditions*, *concurrent update problems*) (vgl. Martin, 2018, S. 52).

2.2.3. Reactive Programming

Reactive Programming ist eine Weiterführung der Konzepte im funktionalen Paradigma. Durch den reaktiven Ansatz bietet es Vorteile beim Entwickeln der Architektur einer Software. Vor allem im Kontext Android kann das event-basierte Reagieren auf Nutzereingaben oder Rückmeldungen verschiedener Datenquellen durch Netzwerkanfragen von Vorteil sein. Darüber hinaus bietet es durch Datenflüsse einen Output-Port (vgl. Abschnitt 2.3.4), an den Daten gesendet werden können, ohne Regeln zur Abhängigkeit verletzen zu müssen.

Der Fokus wird auf Datenflüsse und das Benachrichtigen über Änderungen der Daten gelegt (vgl. Maglie, 2016, S. 4). Unter dem Begriff *Datenflüsse* ist hierbei weniger der Ablauf verschiedenener Datentransformationen in einem Datenflussdiagramm zu verstehen, sondern vielmehr eine Sequenz von Events (API-Server-Antworten, mehrere Nutzereingaben usw.). In der Praxis gibt es meist eine Quelle, die solch eine Sequenz von Events generiert und einen oder mehrere Konsumenten, die auf jedes Event *reagieren* können (vgl. ebd., S. 5f).
Carkci (2014) beschreibt Reactive Programming als Teilmenge des gesamten Konzeptes über Datenfluss, obwohl es meist synonym verwendet wird. Es ist jedoch eine Ausprägung dieser Art des Programmierens. Als Vorteile sind unter anderem die einfachere Handhabung von Parallelität und GUI-Events sowie das Vermeiden diverser Callbacks (callback hell) zu nennen (vgl. ebd., S. 5). Darüber hinaus werden die Konzepte und somit auch die Vorteile der funktionalen Programmierung beibehalten (vgl. Abschnitt 2.2.2).

2.2.4. Prolog

Die in der Arbeit entwickelte App stellt eine logische Programmierumgebung (LP) zur Verfügung. Logische Programmierung ist hierbei ein weiteres Programmierparadigma. Als Sprache kommt Prolog, die sicherlich bekannteste logische Programmiersprache zum Einsatz (vgl. Gabbrielli und Martini, 2010, S. 369). Vor allem in der imperativen Programmierung beschreibt der Entwickler prozedural, was das Programm *tun* soll (vgl. König, 1989, S. 14). „In PROLOG dagegen bedeutet Programmieren das *Erstellen einer Wissensbasis*, oft auch *Datenbasis* genannt " (ebd., S. 14). „Eine Datenbasis der LP enthält Fakten und Regeln, mit deren Hilfe der Programmierer den Wirklichkeitsausschnitt [oder auch nach König (ebd.) *Mini-Welt*] modelliert" (Wagenknecht, 2016, S. 211). Dieser deklarative Ansatz basiert auf der Idee, Berechnungen durch Deduktion logischer Ausdrücke zu verarbeiten (vgl. Gabbrielli und Martini, 2010, S. 369).
Prolog benötigt nur drei Arten von Ausdrücken und nutzt *Fakten*, *Regeln* und *Anfragen* (vgl. König, 1989, S. 15). Hierbei wird beabsichtigt, vorliegende Probleme in die Sprache der mathematischen Logik zu übersetzten. Die Frage zu dem vorliegenden Problem kann durch die aufgestellten Formeln beantwortet werden. Es habe sich gezeigt, dass praktische Probleme und Strukturen leicht mit logischen Aussagen zu beschreiben sind. Im Speziellen seien hier die Prädikatenlogik beziehungsweise die Horn-Formeln zu nennen (vgl. Kleine Büning und Schmitgen, 1988, S. 36). "Eine Hornklausel ist eine Klausel mit höchstens einem positiven Literal" (Styczynski, Rudion und Naumann, 2017, S. 73).
Welche Schritte nötigt sind, um von natürlicher Sprache zu Prädikatenlogik und schlussendlich zu Prolog zu gelangen, macht folgendes Beispiel deutlich. Ein Fakt ist eine logische Aussage, die immer wahr ist (Einer-Hornklausel mit einem positiven Literal).

Da gibt es fünf Ecken.

In Prädikatenlogik sieht dieser Fakt wie folgt aus.

$$\exists x(Ecke(x) \wedge F\ddot{u}nf(x))$$

Übersetzt in Prolog ist der Fakt einfach zu beschreiben.

$$ecken(5).$$

Eine passende Regel verdeutlicht das Prinzip (ungeachtet der mathematisch fachlichen Richtigkeit) noch besser.

Wenn es fünf Ecken hat, ist es eine Pyramide.

In Prädikatenlogik sieht diese Regel wie folgt aus.

$$\exists x(Ecken(x) \wedge Fünf(x) \implies Pyramide(x))$$

Übersetzt in Prolog wiederum wird die Regel wie folgt beschrieben.

$$pyramide :- ecken(5).$$

Wenn von der beschriebenen Wissensbasis ausgegangen wird (siehe Quellcode 2.9), kann eine sinnvolle Abfrage *pyramide.* lauten. Der Prolog-Interpreter versucht, diesen Fakt mit dem gegebenen Wissen zu deduzieren. *Pyramide* ist wahr, wenn und solange *ecken*(5) wahr ist. Dies ist durch den Fakt in der ersten Zeile in Quellcode 2.9 gegeben. Das Programm wird *true.* zurückgeben.

Quelltext 2.9.: Prolog Wissensbasis

```
ecken(5).
pyramide:-ecken(5).
```

Die konkrete Prolog-Syntax und das Vorgehen der Unifikation des Interpreters spielen in dieser Arbeit keine vorrangige Rolle. Daher wird auf deren nähere Beschreibung verzichtet. Als weiterführende Literatur ist vor allem Kleine Büning und Schmitgen, 1988 , *PROLOG : Grundlagen und Anwendungen : mit zahlreichen Abbildungen, Tabellen und Programmbeispielen* zu nennen.

2.2.5. Expertensystem

Nach Styczynski, Rudion und Naumann (ebd., S. 73ff) ist die Wissensbasis von Prolog ein Expertensystem. Die Eigenschaften als deklarative Programmiersprache und das Deduzieren von logischen Fakten beziehungsweise Regeln erlauben es dem Programmierer solch ein System aufzubauen. „Ein Expertensystem sollte man sich als ein Programmsystem vorstellen, bei dem die Fachkompetenz von Experten, die sich in einem eng begrenzten Bereich hervorragend auskennen, in einer Wissensbank gebündelt und informationstechnisch-gerecht zur Lösung von Problemen bereitgestellt wird“ (ebd., S. 10). Das Programm wird folglich mit einer Intelligenz versehen, um das vorher angereicherte Wissen abzurufen. Aufgaben eines Expertensystems sind nach Styczynski, Rudion und Naumann (ebd., S. 13) die folgenden.

- Probleme verstehen

- Probleme lösen
- Lösungen erklären und bewerten
- Wissen erwerben und strukturieren

2.3. Design- und Architekturprinzipien

Wo liegt der Unterschied zwischen Design und Architektur einer Software? Laut Martin (2018) gibt es keinen Unterschied. Auch wenn Architektur, im Gegensatz zur detaillierteren Designebene, häufig auf höherer Abstraktionsebene gesehen wird, könne der Blick auf das Gesamtsystem nicht vollständig sein, wenn einer der beiden Aspekte fehle. Des Weiteren gäbe es keine klare Abgrenzung zwischen den beiden. Jede Entscheidung hätte Auswirkungen auf allen Abstraktionsebenen (vgl. ebd., S. 4). Doch welches Ziel verfolgen solche Entscheidungen?

> „The goal of software architecture is to minimize the human resources required to build and maintain the required system." (Robert C. Martin)

2.3.1. SOLID-Prinzipien

Die SOLID-Prinzipien beschreiben, wie die Funktionen und Datenstrukturen in Klassen anzuordnen sind und wie diese Klassen untereinander verbunden sein sollten (vgl. ebd., S. 58). Dabei gibt es die folgenden fünf Prinzipien.

S	SRP	The Single Responsibility Principle
O	OCP	The Open-Closed Principle
L	LSP	The Liskov Substitution Principle
I	ISP	The Interface Segregation Principle
D	DIP	The Dependency Inversion Principle

Diese sind dann heranzuziehen, wenn bestimmte Architektur- bzw. Designentscheidungen getroffen werden, um diese zu begründen.

The Single Responsibility Principle (SRP)

Dieses Prinzip besagt, dass ein Modul (je nach Anwendungsfall: eine Klasse, eine Datei usw.) nur für einen einzelnen Akteur zuständig sein sollte (vgl. ebd., S. 62). „Akteure [...] stehen jeweils stellvertretend für Personen oder Systeme, die mit dem betrachteten System interagieren“ (Pohl und Rupp, 2011, S. 76). Im Kontext des SRP kann die einzelne *Zuständigkeit* (Single Responsibility) mit *Änderungsgrund* umschrieben werden (vgl. Martin, 2003, S. 97). Dies bedeutet, dass das SRP verletzt ist, wenn mehr als ein Akteur identifizierbar ist, der an einer Funktionalität einen Änderungswunsch äußern könnte und dasselbe Modul/Klasse dafür angepasst werden muss.

The Open-Closed Principle (OCP)

> „A software artifact should be open for extension but closed for modification.“
> (Meyer (1988))

Martin (2018) beschreibt das Prinzip an einem Gedankenexperiment. Gegeben sei eine Webanwendung, die Geschäftsdaten auf einer scrollbaren Seite anzeigt. Die negativen Daten werden hierbei rot dargestellt. Eine neue Anforderung besagt, dass der Geschäftsreport in schwarz-weiß ausgedruckt werden soll. Die lange Liste soll auf mehrere Seiten aufgeteilt und durch Kopf- oder Fußzeilen nummeriert werden. Wie viel wird am bestehenden Code geändert? Wird hauptsächlich Code hinzugefügt? (vgl. ebd., S. 70f). Diese Fragen beantworten, wie gut das OCP umgesetzt wurde.

The Liskov Substitution Principle (LSP)

> What is wanted here is something like the following substitution property:
> If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2 then S is a subtype of T. (Liskov (1987))

Paraphrasierend kann das Prinzip auch wie folgt beschrieben werden. Untertypen müssen substituierbar durch ihre Basistypen (Supertypen) sein (vgl. Martin, 2003, S. 111). Folgendes Beispiel zeigt solch eine potentielle Verletzung. Häufig wird Vererbung als *ist ein/eine*-Beziehung dargestellt. Ist es möglich, ein neues Objekt durch die *ist ein/eine*-Beziehung zu einem bestehenden Objekt zu beschreiben, sollte dies durch Vererbung implementiert werden (vgl. Martin, 2003, S. 113). Aus der Mathematik wissen wir, dass

ein Quadrat eine spezielle Form eines Rechtecks ist. Die Beschreibung *Quadrat ist ein Rechteck* ist demnach passend. Eine Modellierung kann wie in Abbildung 2.2 geschehen. Dies ist sinnvoll, falls das LSP nicht verletzt ist. Um hier eine Verletzung zu zeigen, er-

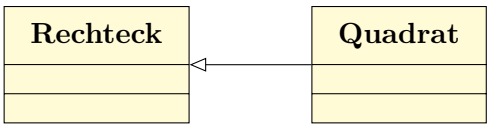


Abbildung 2.2.: Rechteck ist die Basisklasse von Quadrat (vgl. Martin, 2003, S.113)

halten die Klassen Funktionalität (siehe Abbildung 2.3). Die Funktionen des Rechtecks

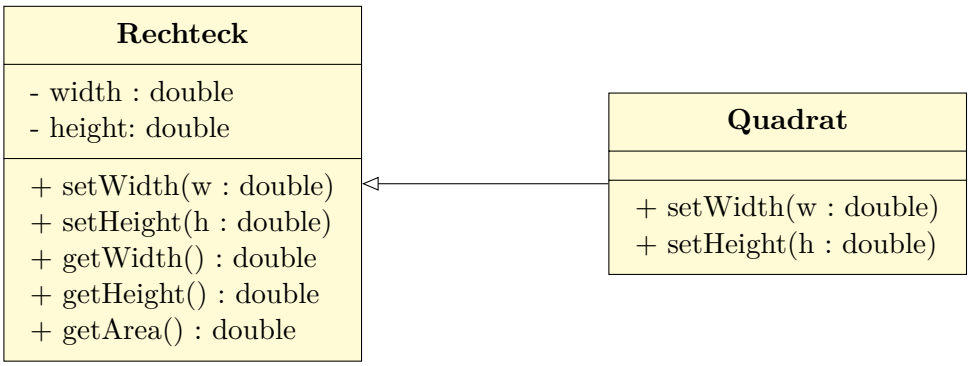


Abbildung 2.3.: Rechteck und Quadrat mit Funktionalität

sind simple *Getter* und *Setter* der privaten Attribute. Die *getArea()*-Methode errechnet den Flächeninhalt (siehe Quellcode 2.10).

Quelltext 2.10.: Rechteck: Methode zum Flächeninhalt

```
fun getArea() {
 return width * height
}
```

Die Methoden der Klasse Quadrat *setWidth(h : double)* und *setHeight(h : double)* überschreiben die der Basisklasse, da ein Quadrat immer gleich lange Seiten hat (siehe Quellcode 2.11).

Quelltext 2.11.: Quadrat: Überschreibt die Setter-Methoden

```
fun setWidth(w : double) {
 width = w
 height = w
}
```

```
fun setHeight(h : double) {
 width = h
 height = h
}
```

Der Zugriff auf Rechtecke und deren Untertypen erfolgt durch die Basisklasse (Beispiel beim Iterieren durch eine Liste). Angenommen das Rechteck wird in einer grafischen Benutzeroberfläche verändert. Dann führt die zuständige Methode einer anderen Klassen folgende Operationen aus (siehe Quellcode 2.12).

Quelltext 2.12.: Verändern der Größe eines Rechtecks

```
fun stretch(r : Rechteck) {
 setWidth(4)
 setHeight(3)
 assert(r.getArea() == 12)
}
```

Solange das übergebene Objekt (r) ein Rechteck ist, funktioniert dies wie erwartet. Wird allerdings die Funktion auf ein Quadrat angewendet, so ist der Flächeninhalt 9 und nicht 12. Somit kann dieser Umstand der Definition der Liskovschen Substitution nicht genügen.

The Interface Segregation Principle (ISP)

Das ISP nimmt sich des Problems zu großer Interfaces (vgl. ebd., S. 135) an, auf die verschiedene Nutzer zugreifen (vgl. Abbildung 2.4). Martin zeigt dies anhand zweier Klassendiagramme. Die Lösung ist, für die jeweiligen Funktionalitäten separate Interfaces bereit zu stellen (siehe Abbildung 2.5). Das ISP sei auf unterster Ebene eher ein Sprach- als ein Architekturproblem (vgl. Martin, 2018, S. 85). Dynamisch typisierte Sprachen hätten im Falle des ISP keine Probleme, da ein erneutes Kompilieren bei Änderung der externen Ressource nicht notwendig wäre. Abhängigkeiten werden während der Laufzeit aufgelöst. Projektteams, die Anwendungen in mehreren Teams mit statisch typisierten Sprachen entwickeln, stehen vor solch einer Herausforderung. Aus der Architektursicht mache es allerdings bei allen Sprachen (dynamisch und statisch typisiert) Sinn, das Problem zu betrachten. Auf Modulebene sei zu bedenken, wie welche Klassen in welchem Modul untergebracht werden (vgl. Martin, 2018, S. 104ff). Im Allgemeinen

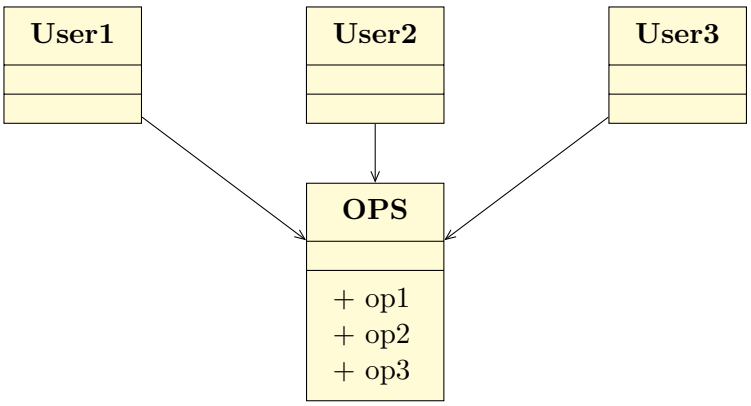


Abbildung 2.4.: Klassendiagramm zum ISP - Problemstellung nach Martin (2018, S. 84)

ist es zu vermeiden, Abhängigkeiten zu anderen Modulen zu haben, die nicht verwendet werden. Dies ist auf Quellcodeebene offensichtlich, da Änderungen in den Modulen unnötige Kompilierungen und Neuauslieferungen nach sich ziehen. Aber auch auf Modulebene ist dies einfach an folgendem Beispiel gezeigt (siehe Abbildung 2.6) (vgl. ebd., S. 86). Angenommen D enthält eine Funktionalität, die F nicht verwendet. S hat demnach keine Kenntnis davon. Änderungen dieser Funktionalität in D bedingen trotzdem eine neue Version von F und demnach auch von S (vgl. ebd., S. 86).

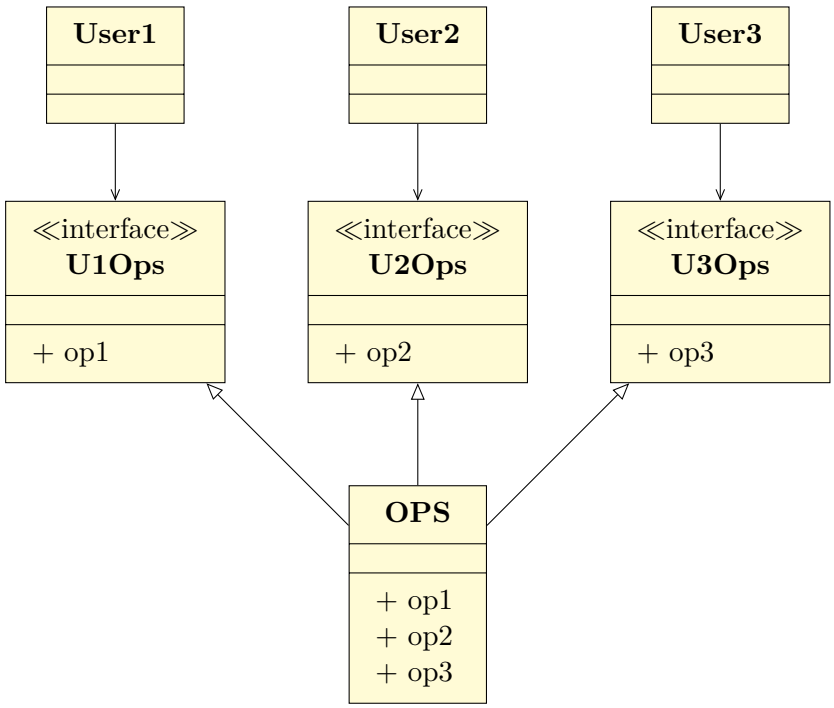


Abbildung 2.5.: Klassendiagramm zum ISP - Lösung nach Martin (2018, S. 85)

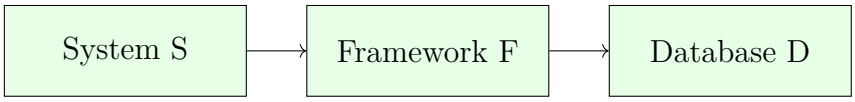


Abbildung 2.6.: Problematische Architektur nach Martin (2018, S. 86)

The Dependency Inversion Principle (DIP)

Das DIP sagt aus, dass die flexibelsten Systeme die sind, bei denen möglichst alle Quellcodeabhängigkeiten auf abstrakte Quellen verweisen und nie auf konkrete Implementierungen (vgl. ebd., S. 87). Das folgende Beispiel von Martin (vgl. 2003, S. 130f) zeigt dies anschaulich. Gegeben sei eine Button-Klasse, die durch eine *Poll*-Nachricht angesprochen wird. Dies kann ein Hardwareschalter oder Softwarebutton sein. Abhängig vom Zustand soll eine Lampe in Folge der *Poll*-Nachricht an- oder ausgeschaltet werden. Abbildung 2.7 zeigt eine mögliche Lösung, die zwar naheliegend, aber nicht sinnvoll ist. Ein

Abbildung 2.7.: Einfaches Modell des Buttons und der Lampe (vgl. Martin, 2003, S. 130)

Blick in den Quellcode der Button-Klasse 2.13 zeigt die starke Verbindung des Buttons

mit der Lampe durch die direkte Abhängigkeit des konkreten Objektes.

Quelltext 2.13.: Button - Implementierung der Lampe

```
class Button{
 private val lamp = Lamp()
 fun poll(){
  if (/*check state*/){
   lamp.turnOff()
  }else{
   lamp.turnOn()
  }
 }
}
```

Diese Abhängigkeit bedeutet, dass der Button von Änderungen der Lampe direkt betroffen ist. Darüber hinaus ist es nicht möglich, den Button wiederzuverwenden. Die Steuerung eines Motorobjekts wäre so nicht möglich. Die Abstraktion wurde nicht von den konkreten Implementierungsdetails getrennt. Um diese zu erreichen, muss die zu Grunde liegende Abstraktion identifiziert werden. Am Beispiel des Buttons und der Lampe ist der Anwendungsfall das Bedienen der Lampe. Abbildung 2.8 zeigt eine Alternative. Durch die Einführung des Interfaces *SwitchableDevice* ruft der Button lediglich

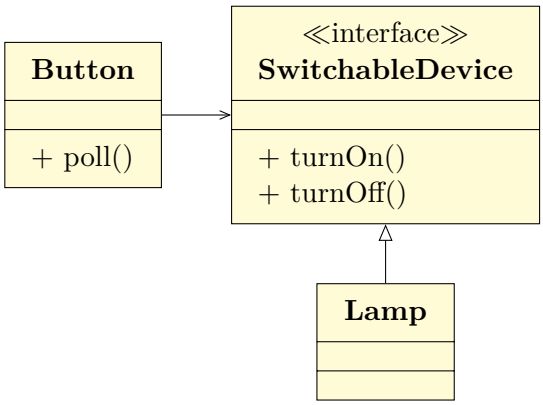


Abbildung 2.8.: DIP am Beispiel der Lampe (vgl. Martin, 2003, S. 131)

abstrakte Methoden auf, um die Lampe zu steuern. Dies erhöht die Flexibilität um ein Vielfaches. Jede Klasse, die dieses Interface implementiert, kann mit Hilfe des Buttons gesteuert werden. Umgekehrt können andere Objekte, die durch das Interface gesteuert werden möchten, dieses implementieren. Hierbei ist es irrelevant, ob ein Button das Interface verwendet oder beispielsweise eine *CheckBox*.

2.3.2. Domain Driven Design

Das Prinzip des Domain Driven Designs (DDD) geht in viele Architektur- und Designüberlegungen mit ein, so auch in die in der Arbeit verwendete Clean Architecture (siehe Abschnitt 2.3.4). Besonders das Repository-Pattern, das eine zentrale Rolle beim Verbinden der Domain und Datenschicht im DDD spielt (vgl. Haywood, 2009, S. 302), bildet eine wichtige Kommunikationsgrundlage dieser Arbeit.

Der domänen-zentrierte Ansatz einer Software im Domain Driven Design ist durch zwei grundsätzliche Ideen gekennzeichnet. Zum einen eine zugängliche, alltägliche Sprache (Ubiquitous Language) und zum anderen deren Abbildung der Domäne im Quelltext (vgl. ebd., S. 4).
Der Begriff *Ubiquitous Language*[3] wurde im Zusammenhang mit DDD von Evans (2004) geprägt. Dies sei aus folgendem Grund wichtig.

> „Domain experts should object to terms or structures that are awkward or inadequate to convey domain understanding; developers should watch for ambiguity or inconsistency that will trip up design“ (ebd., S. 27).

Durch den modellbasierten Ansatz könne ein Modell [der Domäne und der Software] erstellt und genutzt werden, das durch die Kombination einfacher Elemente vollständig und verständlich komplexe Ideen ausdrückt (vgl. ebd., S. 26).
Das Domänenmodell (Domain Model) ist im DDD nicht das naheliegende UML-Diagramm, das eher eine visuelle Repräsentation ausgewählter Aspekte des Modells beschreibt, sondern vielmehr eine Gruppierung verwandter Konzepte, die es zu identifizieren, benennen und deren Beziehungen es zu definieren gilt (vgl. Haywood, 2009, S. 7). Ghosh (2017) macht dies am Beispiel eines Bankkontos deutlich. Innerhalb der Domäne *Bank* hat er vier Aspekte aufgegriffen, die ein Domänenmodell auszeichnen (vgl. ebd., S. 3).

- Es sollten Objekte verwendet werden, die in der Fachdomäne vorkommen. Zum Beispiel: Banken, Konten, Transaktionen.
- Interaktionen zwischen den Objekten beschreiben das Verhalten. Zum Beispiel wird das Konto *belastet*.
- Die Sprache der Domäne muss abgebildet werden (Ubiquitous Language). *Belasten*, *Guthaben*, *Wertpapiere* usw. sind Teil der Domäne.

[3] Ubiquitäre Sprache - allgemeine oder alltägliche Sprache

- Der Kontext, in dem die Interaktionen zwischen den Objekten in der Domäne stattfinden, spiele eine Rolle. Annahmen und Beschränkungen beschrieben das Problem. Ein persönliches Bankkonto kann beispielsweise nur von einer natürlichen Person eröffnet werden.

Im Domain Driven Design existieren einige Entwurfsmuster, die für das Erreichen der Ziele des DDD hilfreich sind. Im Folgenden wird eines herausgegriffen und vorgestellt. Das *Repository*-Pattern spielt eine wichtige Rolle im weiteren Verlauf der Arbeit, da es für die Umsetzung der Clean Architecture essentiell ist (siehe Abschnitt 2.3.4 und 5.4.2). Das Repository ist im Sinne des DDD eine Kombination aus einer Factory und einer Repository-Funktionalität. Eine Factory trennt durch das Bereitstellen einer Schnittstelle das Erstellen konkreter Objekte von der aufrufenden Klasse. Dadurch entkoppelt es die Models von den konkreten Klassen. Repositories stellen eine Schnittstelle bereit, um die Referenzen zu konkreten, erstellten Objekten zu erhalten (vgl. Haywood, 2009, S. 32). Da diese beiden Funktionen jedoch hinsichtlich einer Verantwortlichkeit sehr ähnlich sind, werden sie kombiniert und man spricht von einem Repository. Konkret sollte ein Repository als Interface implementiert sein, um dem Dependency Inversion Principle (siehe Abschnitt 2.3.1) gerecht zu werden. Auch wenn Haywood (vgl. ebd., S. 32f und S. 302ff) das Repository als Vermittler zwischen Domain und Datenbank sieht, ist der konkrete DataStore ein Implementierungsdetail. Demnach ist es irrelevant, aus welcher Quelle die Daten, die durch das Repository bereit gestellt werden, kommen.

2.3.3. Model View Presenter

Um Architekturentscheidungen bezüglich der Oberfläche zu verstehen, wird im Folgenden das Model View Presenter (MVP) Architekturmuster eingeführt. Es beschreibt den Aufbau und die Abhängigkeiten der drei Komponenten *Model*, *View* und *Presenter* für die Darstellung einer Nutzerschnittstelle. Dieses Entwurfmuster hat das Ziel, die View von den Daten (Model) zu trennen. Der Presenter wird zum Vermittler zwischen der View und dem Model (vgl. Mew, 2016, S. 293).
Die Model-Komponente enthält die Daten und die Businesslogik. Der Presenter steuert das Model, um es der View-Komponente zur Verfügung zu stellen. Das Model kennt aber weder den Presenter noch die View. Das bedeutet, dass es keine Abhängigkeiten zu den beiden anderen Komponenten besitzt (vgl. Gharbi et al., 2017, S. 79). Die View-Komponente ist die Präsentationsschicht und enhält keine Logik, sondern lediglich Funktionen, die für das Interagieren mit dem Benutzer nötig sind (Eingaben und die Darstellung) (vgl. ebd., S. 79). Der Presenter steuert den logischen Ablauf zwischen der

View und dem Model. Er nimmt Eingaben aus der View entgegen, ruft die Businesslogik im Model auf und setzt die Daten aus dem Model auf die View (vgl. ebd., S. 80). Abbildung 2.9 verdeutlicht die Abhängigkeiten in der einfachsten MVP-Version.

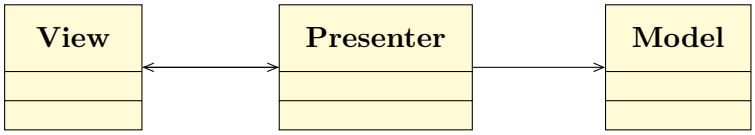


Abbildung 2.9.: Model View Presenter - Abhängigkeiten

Model View Presenter im Android Kontext

Bei der Adaption des MVP-Prinzips auf Android müssen android-spezifische Voraussetzungen der View bedacht werden. Die View bei Android besteht, sofern nicht alle Layoutkonfigurationen programmiert sind, neben einer Activity aus XML-Dateien. Darüber hinaus wird die View neben den Eingaben des Nutzers durch das System gesteuert (vgl. Abschnitt 2.1.2). Somit muss beachtet werden, dass diese Events ebenfalls durch den Presenter zu behandeln sind, falls sie Auswirkungen auf Datenaktionen haben. Ein Beispiel ist das Persistieren von Daten beim Wechseln der Orientierung des Android-Gerätes.
Eine nicht nur android-spezifische, aber sinnvolle Erweiterung ist das Einführen eines View-Interfaces. Der Presenter sollte nicht direkt von der Activity abhängen, sondern durch das Interface von der konkreten Implementierung entkoppelt werden (vgl. Abschnitt 2.3.1). Eine weitere Ergänzung ist ein Presenter-Interface. Durch die Kombination der beiden Interfaces ist ein *Contract* entstanden. Ob solch ein Contract durch das zusätzliche Presenter-Interface sinnvoll ist oder nicht, wird diskutiert (siehe hierzu unter anderem Cervone (2017) und Sanchez (2016)). In der Arbeit wird die Argumentation von Cervone (2017) unterstützt. Man implementiere nicht lediglich ein Interface, sondern erschaffe einen Vertrag zwischen der View und dem Presenter, dessen Interaktionen untereinander auf einen Blick deutlich werden. Darüber hinaus bietet Google in den offiziellen MVP-Beispielen[4] ebenso diese Lösung an. Somit ergibt sich ein angepasstes Bild des MVP-Entwurfmusters, das in Abbildung 2.10 visualisiert ist.

[4] https://github.com/googlesamples/android-architecture/blob/todo-mvp/todoapp/app/src/main/java/com/example/android/architecture/blueprints/todoapp/addedittask/AddEditTaskContract.java

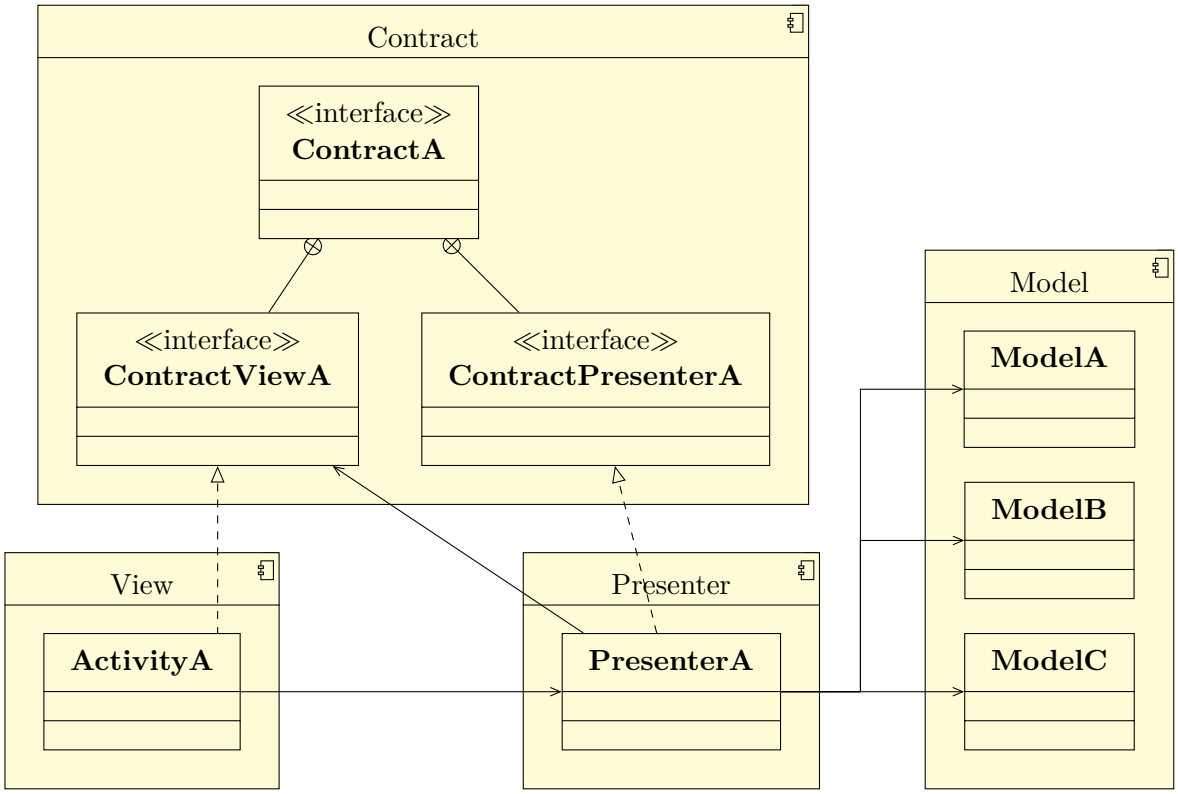


Abbildung 2.10.: Model View Presenter - angepasst in Android

2.3.4. Clean Architecture

Clean Architecture versucht möglichst viele Konzepte und Prinzipien aus *guten* Design- und Architekturprinzipien (vgl. Abschnitt 2.3.1, 2.3.3 und 2.3.2) zusammenzufassen. Es basiert unter anderem auf den grundlegenden Ideen der drei folgenden Ansätze (vgl. Martin, 2018, S. 202).

- Hexagonal Architecture
- Data Context Interaction DCI
- Boundary Control Entity BCE

Im Detail unterscheiden sie sich in einzelnen Architekturüberlegungen, sind jedoch im Groben nahezu identisch (Beispiel: Domain Driven Design Ansatz (siehe 2.3.2)). Die Sicht auf das System ist vergleichbar - über die Trennung der Zuständigkeiten werden die verschiedenen Ebenen der Architektur definiert. Jedes verfügt mindestens über Ebenen für Geschäftsprozesse, Benutzerschnittstelle und Systemschnittstelle. Alle Ansätze erschaffen ein System, das folgende Charakteristika aufweisen soll (vgl. ebd., S. 202).

- Unabhängig vom Framework - „The architecture does not depend on the existence of some library of feature-laden software. This allows you to use such frameworks as tools, rather than having to cram your system into their limited constraints“ (ebd., S. 202).

- *Einfache* Testbarkeit - „The business rules can be tested without the UI, Database, Web Server, or any other external element“ (ebd., S. 202).

- Unabhängig vom User Interface - „The UI can change easily, without changing the rest of the system. A Web UI could be replaced with a console UI, for example, without changing the business rules“ (ebd., S. 202).

- Unabhängig von der Datenbank - „You can swap out Oracle or SQL Server, for Mongo, BigTable, CouchDB, or something else. Your business rules are not bound to the database“ (ebd., S. 202).

- Unabhängig von externen Diensten - „In fact your business rules simply don't know anything at all about the outside world“ (ebd., S. 202).

Die Abbildung 2.11 zeigt ein Architekturkonzept, das alle Ideen aus den genannten Ansätzen und Designprinzipien zusammenführt. Die vier Ringe symbolisieren jeweils verschiedene Ebenen der Software. Die Pfeile, die vom äußersten zum innersten Ring zeigen, beschreiben die Abhängigkeiten. Sie sind *immer* nach innen gerichtet. Daraus ist zu schließen: Je näher eine Ebene am Kern ist, desto höher wird die Abstraktion. Außen befinden sich die Implementierungsdetails zu konkreten Datenbanken, Netzwerkschnittstellen oder Oberflächenframeworks. Der Kern der Software besteht hingegen aus den *Business Rules*. Die *Entities* kapseln die business-spezifischen Regeln einer Software. Dies können reine Objekte mit Methoden und Datenstrukturen sein. Da diese den Kern der ganzen Anwendung bilden, werden sie nicht oder verhältnismäßig selten verändert, da auch keine Änderungen einer anderen Ebene Auswirkungen auf die Entities haben (keine Abhängigkeiten). Die Ebene der *UseCases* beschreibt exakt das, was ihr Name besagt. Die umgesetzten Anwendungsfälle sind in Verbindung mit den Entities abgebildet. Daten von und zu den Entities werden orchestriert und verarbeitet. Durch den integrierten Ansatz des *Domain Driven Designs* (siehe Abschnitt 2.3.2) sollten auch technisch unversierte Stakeholder die Anwendungsfälle alleine an der Namensgebung verstehen. Dies wird als *Screaming Architecture* beschrieben (vgl. ebd., S. 196ff). Die Ebene der *Interface Adapters* ist dafür zuständig, die Daten von den äußeren Quellen für die inneren Kreise in entsprechend passende Form umzuwandeln und umgekehrt. In einer MVC-Struktur wären hier alle Komponenten für eben diese integriert (vgl. ebd.,

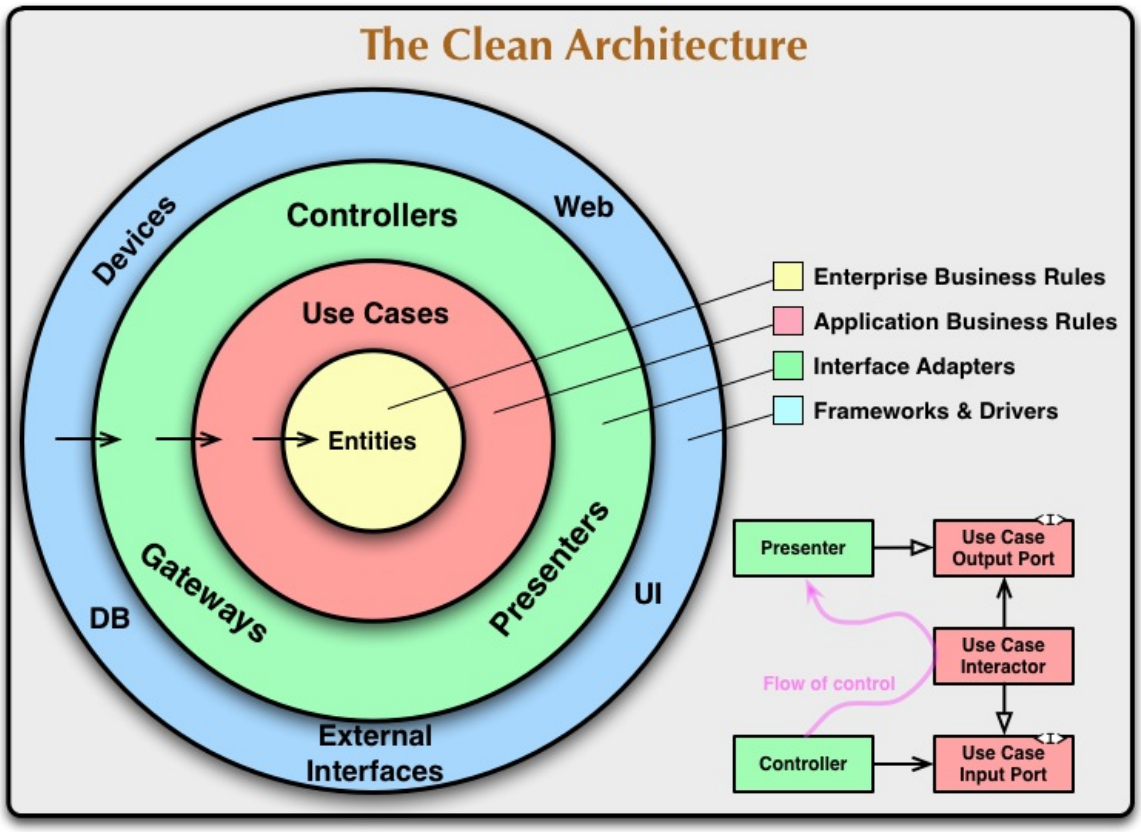


Abbildung 2.11.: *Clean Architecture* Schemadarstellung nach Martin (2012)

S. 205). Die hier enthaltenen Models sind Datenstrukturen, die hin zu den UseCases und von ihnen zurück zum *Presenter* und der *View* geleitet werden. Ebenso sind Daten aus externen Quellen (Web, DB, usw.) in eine davon unabhängige Form zu bringen. Die äußerste Ebene beinhaltet meist Frameworks und externe Schnittstellen. Aus diesem Grund ist hier die Abstraktion am geringsten. An dieser Stelle ist zu implementieren, welche konkrete Datenbank genutzt und welche API im Web angesprochen wird. Eine Änderung der Datenbanktechnologie hätte in dem Fall lediglich auf die Model-Objekte Auswirkungen, die für eine Kommunikation mit der *Interface Adapters*-Ebene genutzt werden.
Die Darstellung in vier Ebenen ist als Beispiel zu sehen, denn abhängig vom zu entwickelnden System können durchaus mehr Ebenen sinnvoll sein. Wichtig ist jedoch das Einhalten der *Dependency Rules*, sodass keine Ebene die weiter äußeren Ebenen direkt verwendet - also von ihnen abhängt.

Nach den *Dependency Rules* ist es nicht möglich, dass der *Presenter* vom *UseCase* Daten durch einen direkten Aufruf erhält. Doch wie kann eine Aktion auf der UI (äußerste Ebene) bis zu den *UseCases* und *Entities* (innerste Ebene) vordringen, um dann verar-

beitet auf der Oberfläche zu erscheinen, ohne dass die UI-Methoden von innen aufgerufen werden? In Abschnitt 2.3.1 ist die Antwort unter *The Dependency Inversion Principle* zu finden. Direkte Aufrufe nach innen gehen an einen Input-Port der nächsten Ebene. Daten aus der nächst inneren Ebene werden durch das Implementieren eines Output-Ports aus dieser Ebene bereitgestellt. Eine Schemadarstellung ist in der rechten Ecke in Abbildung 2.11 zu sehen. Doch welche Daten sollten über diese Grenzen ausgetauscht werden? Hierzu bietet Martin (vgl. 2018, S. 207) eine Erklärung. Typischerweise seien es einfache Datenstrukturen, die diese Grenzen passieren. Darüber hinaus seien simple Datentransferobjekte oder auch Methodenargumente möglich. Wichtig sei, dass diese Daten isoliert zum Implementierungsdetail aufgebaut sind und möglichst nur genau die Informationen beinhalten, die benötigt werden. Ein UseCase sollte beispielsweise keine Datenbankobjekte verarbeiten, da eine Änderung der Datenbank somit direkten Einfluss auf alle anderen Ebenen hätte. Die *Dependency Rules* wären verletzt. Die Daten sollten in einer Form aufbereitet werden, die möglichst passend für die nächst innere Ebene ist.

2.4. Softwarequalität

Wenn die Qualität von Softwaresystemen beschrieben wird, sollte auf die offiziellen ISO-Normen für Softwarequalität verwiesen werden, die zahlreich vorhanden sind. Diese beziehen sich nicht immer direkt auf die Softwarequalität, hängen jedoch teilweise stark mit dieser zusammen. So können Normen wie *ISO 9241* zur Ergonomie der Mensch-System-Interaktion genauso Einfluss auf die Softwarequalität haben wie der Entwicklungsprozess und somit die Prozessqualität beziehungsweise ein gewisser Reifegrad (Beispiel *SPICE*). Um jedoch konkret Softwarequalitätskriterien hinsichtlich eines Softwareproduktes zu identifizieren, ist die Norm *ISO/IEC 25010* (ISO/IEC, 2011) (ehemals *DIN ISO 9126*) heranzuziehen. Sie orientiert sich stark an ihrem Vorgänger *DIN ISO 9126* und ist, teils neu strukturiert und um einige Abschnitte erweitert, 2011 veröffentlicht worden (vgl. Wagner, 2013, S. 60).
Nach Wagner beschreibe das *Product Quality Model* die wichtigsten Produktqualitätseigenschaften für Software, die im Folgenden erläutert werden. In Anhang A.16 sind die Eigenschaften mit dazugehörigen Unterkategorien dargestellt. *Functional suitability* befasst sich damit, ob das Produkt die funktionalen Bedürfnisse und Anforderungen der Nutzer und Kunden abbildet. *Reliability* sei hingegen für viele ein typisches Synonym für Softwarequalität (vgl. ebd., S. 62), da die Fehlertoleranz des Systems und die Verfügbarkeit (neben *Performance efficiency*) den direktesten Einfluss auf das Nutzererlebnis

haben. *Performance efficiency* beschreibt, wie schnell die Anwendung auf die Anfrage beziehungsweise Eingabe eines Nutzers reagiert und dessen Aktionen verarbeitet. *Compatibility* definiert die Qualität eines Produktes, sofern der Einsatz keine anderen Systeme stört und eine Zusammenarbeit mit diesen ermöglicht. Die Eigenschaft *Usability* umfasst alle Aspekte, die mit der einfachen Benutzung zusammenhängen. Dies hat häufig Auswirkungen auf eine intuitive Benutzeroberfläche und steile Lernkurven. Der Aspekt der *Security* ist besonders bei Mehrnutzersystemen und Anwendungen, die über das Netzwerk erreichbar sind wichtig - dies trifft auf die meisten Systeme zu (vgl. Wagner, 2013, S. 63). Eingaben und Zugriffe sollen autorisiert, verfolgbar, unveränderbar (durch Dritte) und authentifiziert erfolgen. Eine Produkteigenschaft, die eher entwicklerspezifisch einzuordnen ist, heißt *Maintainability*. Sie beschreibt, ob ein System so entworfen ist, dass zukünftige Veränderungen, Anpassungen und Erweiterungen effizient eingearbeitet werden können. Der achte und letzte Aspekt *Portability* ist erneut entwicklerspezifisch und vor allem dann von Interesse, wenn das Produkt auf verschiedenen Plattformen lauffähig sein soll oder unerfahrene Endnutzer die Software selbstständig installieren.

Ist es das Ziel für eine Software höchster Qualitätsansprüche, alle Kriterien bestmöglich abzudecken und zu erfüllen? Ist dies überhaupt möglich? „Am Beispiel der Kriterien Effizienz und Übertragbarkeit wird schnell deutlich, dass dieses Vorhaben von Grund auf zum Scheitern verurteilt ist“ (Hoffmann, 2008, S. 10). Hoffmann führt weiter aus, dass Programme, die eine hohe Übertragbarkeit (Portabilität) haben, in der Regel auf einem abstrakten Niveau geschrieben sind. Teilweise mit Frameworks, die den kleinsten gemeinsamen Nenner bilden. Hier wird schnell klar, dass diese Anwendungen hinsichtlich der *Effizienz* nicht an eine Software heranreichen können, die auf eine Plattform oder Hardwarekonfiguration abgestimmt ist (vgl. ebd., S. 10f). Somit kommt es auf das Projekt und den Kontext an, welche Qualitätskriterien erhöhte Aufmerksamkeit erhalten sollten.

2.5. Natürliche Sprache verarbeiten - NLP

Die Anwendung, die im Rahmen dieser Arbeit entsteht, soll das Programmieren mit Hilfe natürlicher Sprache ermöglichen. Folglich ist das korrekte Verstehen der sprachlichen Eingaben eines Nutzers eine Voraussetzung. *Natural language processing* (NLP)

beschäftigt sich mit der Verarbeitung natürlicher Sprache. Genauer mit der automatischen Verarbeitung menschlicher Sprache (vgl. Dale, Moisl und Somers, 2000, S. 1). Traditionell bezieht sich das Arbeiten in diesem Feld auf das Analysieren der Sprache in drei Kategorien. Um ein möglichst präzises Bild der Linguistik wiederzugeben, werden eben diese Kategorien der Sprachwissenschaften zur Analyse verwendet. In mehreren Schritten werden Syntax (Abfolge der Wörter), Semantik (Bedeutung der Wörter) und Pragmatik (situative, kontextabhängige Deutung der Wörter) verarbeitet.
NLP ist ein signifikantes Anwendungsgebiet der künstlichen Intelligenz. Computer würden als intelligent gelten, wenn sie, anstatt formaler Befehle wie C, FORTRAN oder PASCAL, natürliche Sprache als Eingabe hätten. Solche Kommunikationsmöglichkeiten mache die Interaktion mit Computern um ein Vielfaches einfacher (vgl. Kumar, 2011, S. 2).
Die Idee, mit natürlicher Sprache zu programmieren, reicht bis in die 1970er Jahre zurück (vgl. Landhaeusser, 2016, S. 47). Einen der ersten Ansätze lieferte ein System von Ballard und Biermann (1979), das zwei Matrizen multipliziert (vgl. Landhaeusser, 2016, S. 47). Eine natürlichsprachliche Möglichkeit, die sehr nah an einer konkreten Programmiersprache ist, wurde von Price et al. (2000) entwickelt und übersetzt die natürliche Sprache in konkrete Java Klassen, Methoden und Parameter. Hierbei muss der Nutzer die jeweils zu adressierende Klasse immer erneut ansprechen. SmartSynth ist eine Schnittstelle zur Beschreibung von automatisch ausführbaren Skripten, die von Le, Gulwani und Su (2013) in einem Paper vorgestellt wurden. In SmartSynth können Regeln und Ereignisse definiert werden, die die Bedienung im Auto betreffen.
Da ein Vergleich oder eine wissenschaftliche Forschung zum Thema NLP oder Computerlinguistik nicht Fokus dieser Arbeit ist, wird auf die verschiedenen Arten mit Vor- und Nachteilen nicht weiter eingegangen. Das in diesem Rahmen verwendete Dialogflow kapselt viele der NLP-Konzepte auf einer für den Entwickler abstrakten Ebene.

2.5.1. Dialogflow

Dialogflow[5] ist ein Dienst von Google, mit dessen Hilfe Entwickler digitale Assistenten erstellen können. Er bietet eine modellierbare NLP-Schnittstelle. Google stellt zum Modellieren des Assistenten online eine grafische Nutzeroberfläche bereit. Die aus dem Modell resultierende Service-Instanz läuft auf den Servern von Google. Eine Authentifikation erfolgt über Client-Schlüssel, die in jeder Instanz generiert werden. Folglich

[5] ehemals API.AI

müssen Clients, die diese Assistentenschnittstelle nutzen möchten, über eine Internetanbindung verfügen. Die Offline-Nutzung ist nicht möglich. Im Folgenden sind die Konzepte von Dialogflow dargestellt, die im Rahmen dieser Arbeit verwendet werden. Für weiterführende Informationen sind die Projektwebseite[6], sowie die Video-Tutorials[7] von Ido Green[8] auf Youtube zu nennen.

Intents

Auf Grundlage erstellter Beispielsätze leitet der Dienst vorher definierte Intents ab (*ACHTUNG:* Diese Intents in Dialogflow haben nichts mit den Android Intents aus Abschnitt 2.1.1 gemein). Intents sind Absichten, die der Nutzer bei der Interaktion mit dem System hat. Sie unterscheiden sich je nach Anwendungsfall. Ein Assistent für Essensbestellungen könnte folgende Intents erhalten.

1. *start_order*
2. *cancel_order*
3. *confirm_order*

Ein Beispielsatz für den ersten Intent (*start_order*) könnte wie folgt lauten. *Ich möchte eine Pizza bestellen* (siehe Abbildung 2.12).

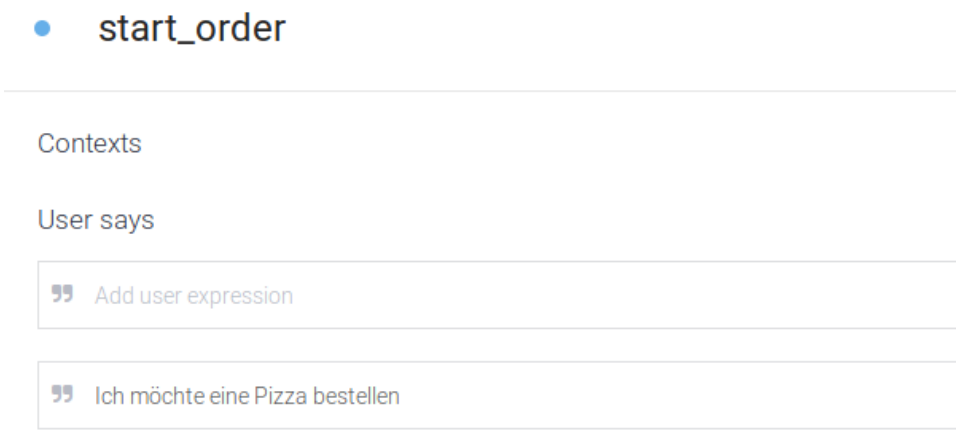


Abbildung 2.12.: Beispielsatz des Intents *start_order*

[6] https://dialogflow.com/
[7] https://www.youtube.com/watch?v=3NIopUsI4ik&list=PLOU2XLYxmsILvfJcIASBDbgfxloFz_XsU
[8] Google Mitarbeiter - Vortragender zu Dialogflow auf Google Developer

Sobald nun ein Nutzer diesen Satz (oder einen ähnlichen) sagt, weiß Dialogflow, dass der Nutzer eine Bestellung aufgeben möchte (siehe Quellcode 2.14 Zeile 3 und 12).

Quelltext 2.14.: JSON Ausschnitt aus der Antwort der Dialogflow-Schnittstelle - Intent erkannt explizit

```
1  "result": {
2          "source": "agent",
3          "resolvedQuery": "Ich möchte eine Pizza bestellen",
4          "action": "",
5          "actionIncomplete": false,
6          "parameters": {},
7          "contexts": [],
8          "metadata": {
9                  "intentId":
                    ↪ "4ab19376-f32d-4aff-be9f-5b104dfa9ab2",
10                 "webhookUsed": "false",
11                 "webhookForSlotFillingUsed": "false",
12                 "intentName": "start_order"
13         },
```

Interessant ist hierbei, dass es nicht nötig ist, dass ein Nutzer exakt diesen Satz verwendet. *Hallo ich möchte bitte eine Pizza bestellen* wird ebenfalls ohne zusätzlichen Aufwand als Bestellwunsch erkannt (siehe Quellcode 2.15 Zeile 3 und 12).

Quelltext 2.15.: JSON Ausschnitt aus der Antwort der Dialogflow-Schnittstelle - Intent erkannt implizit

```
1  "result": {
2          "source": "agent",
3          "resolvedQuery": "Hallo ich möchte bitte eine Pizza
            ↪ bestellen",
4          "action": "",
5          "actionIncomplete": false,
6          "parameters": {},
7          "contexts": [],
8          "metadata": {
9                  "intentId":
                    ↪ "4ab19376-f32d-4aff-be9f-5b104dfa9ab2",
```

```
10                "webhookUsed": "false",
11                "webhookForSlotFillingUsed": "false",
12                "intentName": "start_order"
13            },
```

Entities

Entitäten sind Begriffe oder Begriffszusammenhänge, die semantisch erfasst werden. Dies können Zeitpunkte, Orte oder wie im Beispiel der Bestellung Produktkategorien sein. Damit können bestimmte, semantische Informationen aus dem Intent identifiziert werden. Abbildung 2.13 zeigt Produktkategorien (*product_category*). Dies kann hilfreich

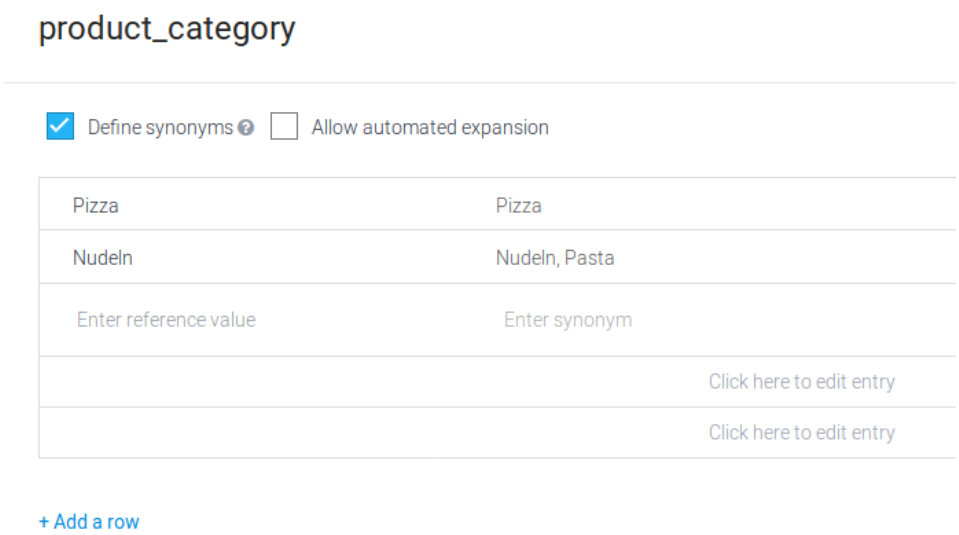


Abbildung 2.13.: Entität *product_category*

sein, um dem Nutzer alle verfügbaren Produkte einer bestimmten Kategorie zur Verfügung zu stellen. Neben den konkreten Werten, die zu der Entität gehören, können Synonyme verwendet werden. *Pasta* zählt nach der Modellierung genauso zur Produktkategorie *Nudeln* wie der Begriff selbst. Eine weitere sinnvolle Entität könnte ergänzt werden. *product_name* enthält folglich konkrete Produkte.
Intents erkennen diese Entitäten. Hierzu muss ein Feld der entsprechenden Entität hinzugefügt werden (siehe Abbildung 2.14). An dieser Stelle können ein beliebiger Parameter-Name sowie die entsprechende Entität gewählt werden. Zusätzlich besteht die Möglichkeit, dieses Feld als *erforderlich* zu markieren, um explizit danach zu fragen, falls der

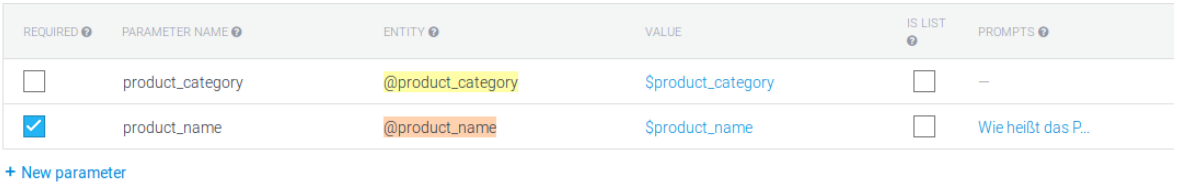


Abbildung 2.14.: Intent Parameter

Nutzer es nicht angibt. Um etwas zu bestellen, ist in diesem Beispiel der Produktname erforderlich. Das Nachfragen kann direkt unter dem Punkt PROMPTS eingefügt werden. *Wie heißt das Produkt, das du bestellen willst?*
Begriffe aus den Beispielsätzen (siehe Abschnitt Intents), auf die die Parameter passen, sind mit der entsprechenden Farbe hinterlegt (siehe Abbildung 2.15). Eine Anfrage

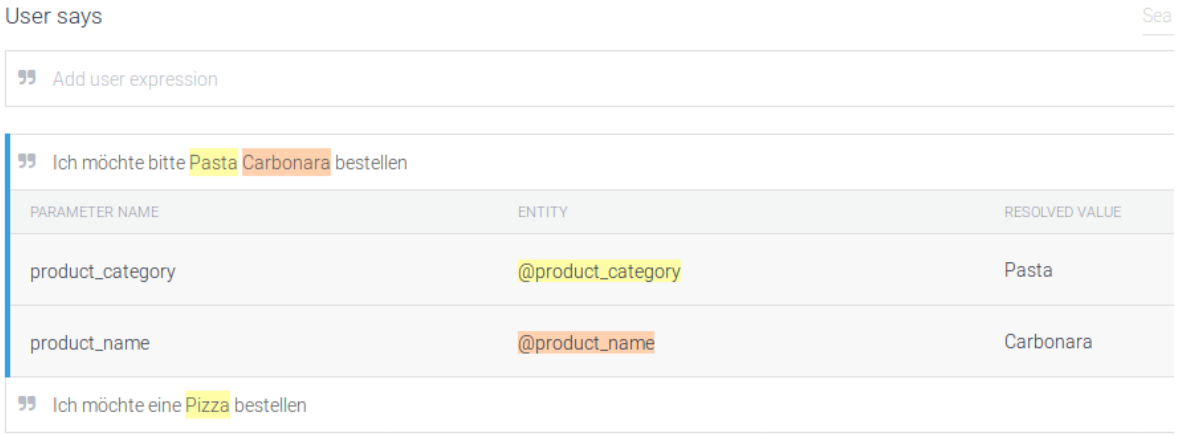


Abbildung 2.15.: Beispielsätze mit erkannten Parametern

an die Schnittstelle mit dem Inhalt *Ich möchte bitte Pasta Carbonara bestellen* (siehe Quellcode 2.16 Zeile 3), liefert eine korrekte Identifikation des Intents (Zeile 15), der Produktkategorie (Zeile 7) und des Produktnamens (Zeile 8).

Quelltext 2.16.: JSON Ausschnitt aus der Antwort der Dialogflow-Schnittstelle - Intent, Kategorie und Name

```
1 "result": {
2         "source": "agent",
3         "resolvedQuery": "Ich möchte bitte Pasta Carbonara
          ↪ bestellen",
4         "action": "",
5         "actionIncomplete": false,
6         "parameters": {
```

```
7                 "product_category": "Nudeln",
8                 "product_name": "Carbonara"
9         },
10        "contexts": [],
11        "metadata": {
12                "intentId":
                   ↪ "4ab19376-f32d-4aff-be9f-5b104dfa9ab2",
13                "webhookUsed": "false",
14                "webhookForSlotFillingUsed": "false",
15                "intentName": "start_order"
16        },
```

Example- und Template-Modus

Neben der gezeigten Möglichkeit konkrete Beispielsätze (Example mode) zu nutzen (*Ich möchte eine Pizza bestellen*), bietet Dialogflow den Template-Modus an. Hierbei werden die konkreten Parameter in den Sätzen (*Pizza*, *Pasta*, *Carbonara*) durch die abstrakten Entitäten ersetzt (siehe Abbildung 2.16). Bei einer Datenmenge (Beispielsätzen, Entitä-

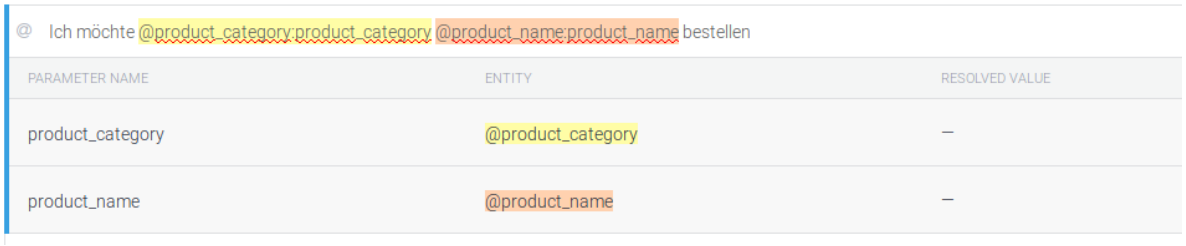


Abbildung 2.16.: Dialogflow Template-Modus

ten etc.), die groß genug ist, nutzt Dialogflow dieses Template, um Produktkategorien und Produktnamen zu erkennen, die nicht in den konkreten Entitäten genannt sind. Dadurch eröffnet sich für den Entwickler eine immense Flexibilität, da so auch Szenarien verstanden und erkannt werden, die nicht explizit formuliert wurden. Hierzu verwendet Dialogflow Machine-Learning-Algorithmen. Im Rahmen dieser Arbeit wird maschinelles Lernen lediglich durch den Dienst Dialogflow genutzt und nicht eigens implementiert.

Training

Um Dialogflow beim richtigen Zuordnen der Intents und Entitäten zu unterstützen, wird der Training-Bereich zur Verfügung gestellt. Alle Anfragen an den Dienst werden angezeigt. Händisch kann der Entwickler falsch zugeordnete Intents bzw. Entitäten korrigieren oder richtig zugeordnete bestätigen. Dieses Vorgehen kommt aus dem maschinellen Lernen. Der Algorithmus benötigt geprüfte Trainingsdaten, um Wahrscheinlichkeiten zu berechnen, welcher Intent oder welche Entität gemeint ist.

3. Didaktische Grundlagen

Das didaktische Grundlagenkapitel bildet die Basis, um den Einsatz der im Rahmen der Arbeit entwickelten Software in den schulischen Kontext zu stellen. Es wird zum einen auf die grundlegenden Aspekte der Lehr- und Lerntheorie und zum anderen auf fachdidaktische Überlegungen eingegangen. Diese konzentrieren sich auf die (Digitalen-)Medien und deren didaktischen Einsatz im Mathematikunterricht sowie die informatikdidaktischen Grundüberlegungen. Die mathematische Konzentrierung ist zum Verständnis der resultierenden Unterrichtsstunden wichtig, da die Forschung im fachlichen Rahmen der Mathematik stattfindet. Aus dem Oberbegriff der Pädagogik wird als Teilgebiet die Didaktik herausgegriffen, da diese den größten Einfluss auf die Konzeption der Forschung hat. Weitere Bereiche aus dem Gebiet der Pädagogik wie *Gesprächsführung*, *Classroom-Management* oder *Lehr- und Sozialformen* werden in der Erhebung implizit angewendet, aber hier nicht näher erläutert.

3.1. Wie lernen wir?

Im Kontext von Schule, Hochschule und anderen Bildungsinstitutionen wird der Begriff Lernen gerne als Synonym für den Wissenserwerb genutzt (vgl. Wild und Möller, 2014, S. 4). Daher wird im weiteren Verlauf der Arbeit der Begriff *Lernen* gleichbedeutend mit Wissenserwerb verwendet. Nach Gage, Berliner und Bach (1996) sind unter Lernen alle Aktivitäten zu verstehen, durch die Subjekte ihr Wissen und Können verändern. „Unter psychologischer Perspektive wird Lernen als Aufbau bzw. Umbau von kognitiven Strukturen verstanden“ (Petko und Jürgens, 2014, S. 23). Grundsätzlich wird zwischen implizitem (vgl. Reber, 1996, S. 88ff) und explizitem (vgl. ebd., S. 23) Lernen unterschieden. Petko und Jürgens (2014, S. 24ff) vergleichen dies im Kontext von Medien mit dem Schauen einer Unterhaltungssendung (implizit) und dem bewussten Ansehen eines Dokumentarfilms mit dem *Ziel* sich weiterzubilden. Weiter beschreiben Petko und Jürgens noch zwei zu beachtende Unterschiede. Zum einen die *Tiefe* des Gelernten. Hierbei ist wichtig, ob das Lernen nur oberflächliches Auswendiglernen beinhaltet oder vertieftes

(deep level processing) Verständnis vorhanden ist. Zum anderen kann noch zwischen persönlichen Erfahrungen (Beispiel: das Anfassen einer heißen Herdplatte) oder kulturellen Wissensbeständen (Beispiel: das Schreiben von Buchstaben) differenziert werden.

3.1.1. Lehr- und Lerntheorie

Die Literatur unterscheidet lerntheoretische Positionen häufig in vier abgegrenzte Haupt-Paradigmen, wobei es noch einige weitere angelehnte Ansätze gibt, die für eine grobe Übersicht nicht relevant sind. Die *behavioristische Lehr- und Lerntheorie* beschreibt Lernen als Konditionierung durch Feedback. In dieser sehr geführten, durch Wiederholungen gekennzeichneten Form sind die sogenannten *Drill-and-Practice* Programme wie typische Vokabeltrainer einzuordnen (vgl. Petko und Jürgens, 2014, S. 27). In der *kognitivistischen Lehr- und Lerntheorie* werden Vergleiche zur Computertechnik auf Grund von Annahmen zur Funktionsweise des menschlichen Denkens gezogen. Damit geht die Klassifizierung verschiedenen Wissens (deklarativ, prozedural, metakognitiv uvm.) und deren Verknüpfung untereinander (einzelne Wissenseinheiten vernetzen sich zu größeren Speicher- und Informationseinheiten) einher (vgl. ebd., S. 28ff). Die *konstruktivistische Lehr- und Lerntheorie* sieht Lernen eher als kreativen Prozess jedes einzelnen Individuums. Die individuelle Konstruktion des Wissens durch angebotene Lernumgebungen sollen den Wissenszuwachs ermöglichen. Das problembasierte Lernen und Gestalten steht im Mittelpunkt. Das Wissen solle den Lernenden nicht eingetrichtert, sondern angeboten werden. Somit können sie ihre Erfahrung auf Basis des individuellen Vorwissens ergänzen oder revidieren (vgl. ebd., S. 32ff). Die *sozialkonstruktivistische Lehr- und Lerntheorie* erweitert und widerspricht den konstruktivistischen Ansätzen in dem Punkt, dass Lernen nur ein individueller Prozess ist. „[Lernen] ist ein Mix aus Zusehen, Etwas-erklärt-Bekommen, Mitmachen, Ausprobieren und Neuerfinden, wobei bestehende Wissensbestände nicht nur angeeignet, sondern auch aktualisiert und modifiziert werden. Meinungsunterschiede sind ein wichtiger Faktor im Lernprozess“ (ebd., S. 34). Petko und Jürgens kritisieren stark die zu eng gefassten und als Gegensatz betrachteten einzelnen Ansätze. „Umfassend[e] Lernprozesse [sollen] möglichst vielfältige Herangehensweisen und einen Lerngegenstand mit einer Abfolge unterschiedlicher Lernaktivitäten [beinhalten]“ (ebd., S. 40).

Für den Rahmen der Arbeit gilt dies ebenfalls. Es muss auf einen vielfältigen Zugang der Lernmöglichkeiten geachtet werden. Somit ist es für das Erstellen der Unterrichtsstunden (siehe Kapitel 6) entscheidend, neben der starken Handlungsorientierung des

einzelnen Schülers (konstruktivistisch) das gemeinsame Entdecken bzw. Voneinanderlernen (sozialkonstruktivistisch) sowie auch das Wiederholen (behavioristische - Beispiel: erneutes Sprechen von Fakten und Regeln siehe Abschnitt 6.2) und das Aufzeigen von Zusammenhängen (kognitivistisch) zu integrieren.

3.2. Informatikdidaktik

Ein wesentliches Merkmal von Informatik ist die *Kontextualisierbarkeit*. Hinsichtlich des Unterrichtsfaches Informatik sind die Probleme (meist) nicht fachimmanent, sondern beziehen sich auf andere Anwendungsgebiete (vgl. Modrow und Strecker, 2016, S. 25). Es bedarf einer *Modellierung* und Übersetzung des fachlichen in ein informatikgerechtes Problem. Die Rekontextualisierbarkeit der Lösung, bezogen auf das zu lösende System, ist anschließend zu bewerten, zu diskutieren und einzuordnen. Hier nehme das Modellieren im Gegensatz zu anderen Fächern (Beispiel Mathematik) einen viel höheren Stellenwert ein. So sei der Modellierungsprozess nicht lediglich eine Vorstufe, sondern (Haupt-)Teil der Auseinandersetzung mit dem Problem (vgl. ebd., S. 25). Das Erarbeiten von Ergebnissen in einer Handlungs- und Problemorientierung steht nach konstruktivistischen Leitideen im Vordergrund, sodass der Lernprozess vom Lehrer begleitet stattfinden sollte (vgl. Schubert und Schwill, 2011, S. 91).
Eine problemorientierte Herangehensweise hat noch weitere Vorteile. Durch das Anwenden von Strategien erlernen die Schülerinnen und Schüler, im Gegensatz zum Anwenden eines Werkzeugs, die nötigen Konzepte, um eine Lösung zu modellieren. Es kann vor einer starken Werkzeugorientierung bewahren (vgl. Hubwieser, 2007, S. 96). Ganz ohne Werkzeugorientierung geht es jedoch auch nicht. Hartmann, Näf und Reichert (vgl. 2007, S. 23) beschreiben die Problematik der Balance zwischen dem Konzept- und dem Produktwissen, die beide eine große Rolle spielen. Konzentriere sich ein Lehrer einseitig auf Konzeptwissen, so würde die Handlungsorientierung fehlen (vgl. auch *E-I-S-Prinzip* Abschnitt 3.3). Bei einer Fixierung auf Produktwissen könnten Schülerinnen und Schüler die Modellierung von Lösungen in einem anderen Gebiet nicht übertragen. Tabelle 3.1 zeigt die Wissensarten als Übersicht.

Produktwissen	**Konzeptwissen**
produktbezogen	produktunabhängig
kurzlebig	langlebig
auswendig lernen, wiedergeben	verstehen und einordnen
isolierte Fakten	Zusammenhänge
wenig Transfer möglich	Transfer möglich
konkret	abstrakt

Tabelle 3.1.: Übersicht Produktwissen und Konzeptwissen nach Hartmann, Näf und Reichert (2007, S. 24)

„Konzeptwissen umfasst die längerfristig gültigen, grundlegenden Zusammenhänge eines Sachgebiets. Produktwissen umfasst die Kenntnisse, die zur Bedienung eines konkreten Produktes nötig sind“ (Hartmann, Näf und Reichert, 2007, S. 23). Für die Lehrkraft ist es demnach wichtig, immer eine Verbindung zwischen dem Konzeptwissen und dessen Anwendung auf das jeweilige Produkt für die Schülerinnen und Schüler darzustellen.

Für die Konzeption einer Unterrichtseinheit hat dies zur Folge, dass die SuS[1] zum einen lernen müssen mit der App zu arbeiten (Produktwissen - Werkzeugorientierung). Zum anderen sollen aber die Konzepte der logischen Programmierung (Fakten, Regeln, Wissensbasis) verstanden werden (Konzeptwissen). Dies ist besonders in dieser Arbeit vorrangig, da durch den deklarativen Ansatz einer logischen Programmiersprache und das Reduzieren der Syntax durch natürliche Sprache das Produkt (die App) und die Bedienung aus didaktischer Sicht stark in den Hintergrund rücken.

3.2.1. Kompetenzen für informatische Bildung im Primarbereich

Im Zuge der angestrebten Forschung und des Einsatzes der App stellt sich neben der grundsätzlichen Orientierung auf Konzeptwissen die Frage, welches Konzeptwissen damit gemeint ist. Wissen, das an Schulen vermittelt wird, ist durch Kerncurricula der entsprechenden Fächer und die dazugehörigen Bildungsstandards festgelegt (ausgenommen die Fachkonferenzen, die auf Grundlage des Kerncurriculums den spezifischen Lehrplan für die eigene Schule ausarbeiten). Für die Grundschule im Bereich Informatik existiert bisher kein Kerncurriculum. Am 13. September 2017 wurde ein Dokument für Bildungsstandards im Primarbereich durch die Gesellschaft für Informatik (GI) e.V. verabschiedet, das den Weg zu einem Kerncurriculum für Schüler dieser Klassenstufen

[1] SuS wird im Folgenden als Abkürzung für Schülerinnen und Schüler verwendet

(1-4) eröffnet (Gesellschaft für Informatik (GI) e. V., 2017). Wissen ist in zwei Kompetenzbereiche unterteilt. Die inhaltsbezogenen Kompetenzen und die prozessbezogenen Kompetenzen, die jeweils immer in Kombination zu betrachten sind (vgl. ebd., S. 7ff). Aus letzterem Bereich sind zwei Kompetenzen hervorzuheben.

> **Kommunizieren und Kooperieren**
> „Die Kinder tauschen sich über eigene Denkprozesse oder Vorgehensweisen mit anderen aus. Sie kommunizieren über informatische Gegenstände und Beziehungen in der Umgangssprache und zunehmend auch in der fachgebundenen Sprache mit fachspezifischen Begriffen. Die Kinder kooperieren zur bzw. bei der Bearbeitung informatischer Probleme."
>
> **Darstellen und Interpretieren**
> „Die Kinder stellen eigene Denkprozesse oder Vorgehensweisen angemessen und nachvollziehbar dar. Dies kann sowohl verbal in mündlicher oder in schriftlicher Form als auch durch den Einsatz von Darstellungsformen wie Skizzen, Tabellen, Mind-Maps usw. geschehen. Sie interpretieren unterschiedliche Darstellungen von Sachverhalten."
>
> (Gesellschaft für Informatik (GI) e. V. (ebd., S. 10))

Aus dem Inhaltsbereich finden sich keine konkreten Kompetenzen wieder, die zum logischen oder deklarativen Programmieren hinführen. Im Bereich *Information und Daten* ist folgendes zu lesen: „*Kompetenz* Die Kinder erläutern den Zusammenhang von Daten und Information sowie verschiedene Darstellungsformen für Daten" (ebd., S. 10). Der dazugehörige Bezug zur Informatik wird über Kodierungen gewählt. „Die Informatik entwickelt Kodierungen, um Information in Daten repräsentieren und maschinell verarbeiten zu können" (ebd., S. 10). Im erweiterten Sinne könnte darunter auch das deklarative Aufbereiten von Daten zu Informationen verstanden werden, die einem System zur Auswertung zur Verfügung stehen. Auch wenn die Schwerpunkte der Kompetenzerwartungen der Gesellschaft für Informatik (GI) e. V. eher in den Bereichen binäre Darstellung und Verschlüsselung gelegt werden (vgl. ebd., S. 11f).

3.3. (Digitale-)Medien in der Mathematikdidaktik

> *„Medien sind einerseits kognitive und anderseits kommunikative Werkzeuge zu Verarbeitung, Speicherung und Übermittlung von zeichenhaften Informationen."* ((Petko und Jürgens, 2014))

Medien lassen sich nach Schmidt-Thieme und Weigand (vgl. 2015, S. 462f) nach ihrer Funktion und ihrem Einsatzzweck unterscheiden. Dabei geht es zum einen um die Wirkung und zum anderen um das Ziel, das mit dem Einsatz von Medien erreicht werden soll. Die Medien sollen den aktiven und selbstgesteuerten Prozess des Lernenden unterstützen. Dies spiegele „sich [auch] in der (heutigen) Mediengestaltung für den Unterricht verstärkt wider, wenn Medien in Lernumgebungen [...] integriert werden oder in Form von produktiven Übungsphasen, Selbstlerneinheiten und Erkundungsaufgaben an Bedeutung gewinnen“ (ebd., S. 463). Dabei solle darauf geachtet werden, dass das *E-I-S-Prinzip* auch beim Einsatz der Medien im Mathematikunterricht weiterhin als grundlegend angesehen wird (vgl. ebd.). Das *E-I-S-Prinzip* ist eine Theorie von Jerome Bruner und bezieht sich „auf [drei] verschiedene Darstellungsebenen (Repräsentationsmodi), die in Wechselbeziehung zueinander stehen“ (Storz, 2009, S. 23). So solle jede dieser Ebenen beim Lernen angesprochen werden und sie sollen sich gegenseitig unterstützen. Die ***e**naktive Darstellung* beschreibt das Handeln mit konkretem Material. Die ***i**konische Darstellung* arbeitet mit der Darstellung mit Hilfe von Bildern und Grafiken. Die dritte Ebene ist die ***s**ymbolische Darstellung*, die durch den Zugang über Sprache oder Zeichensysteme angesprochen wird (vgl. ebd., S. 23).
Medien im Mathematikunterricht lassen sich in traditionelle und digitale Medien aufteilen. Im Rahmen dieser Arbeit sind vorrangig die digitalen Medien von Interesse. „Zu den *digitalen (elektronischen) Medien* gehören Computer, Taschenrechner, Präsentationsmedien (Beamer oder interaktive Tafel), Computerprogramme oder digitale Bücher. Im Unterschied zu traditionellen Medien bringt der Einsatz der digitalen Medien die Möglichkeit des schnellen Erzeugens von Darstellungen, der Dynamisierung und des interaktiven Veränderns dieser Darstellungen“ (Schmidt-Thieme und Weigand, 2015, S. 469f).

3.3.1. Einsatz digitaler Technologien (DT)

Nach Petko und Jürgens (vgl. 2014, S. 116) sowie Schmidt-Thieme und Weigand (vgl. 2015, S. 480f) lassen sich die Ideen und der Einsatz digitaler Medien im Unterricht anhand des didaktischen Dreiecks (Lernende, Lehrkräfte und Inhalte) gut verdeutlichen (siehe Abbildung 3.1). Hinsichtlich der Schülerinnen und Schüler steht mit Einsatz digitaler Technologien der Lernzuwachs in Hinblick auf Inhalts- und Prozessziele bzw. erreichter Kompetenzen im Mittelpunkt. „Verstehen Schülerinnen und Schüler beim Einsatz von DT Inhalte - und wenn ja, welche - besser oder zumindest anders? Erwerben sie bessere - oder andere - Prozesskompetenzen? [...] Erwerben sie eine veränderte Ein-

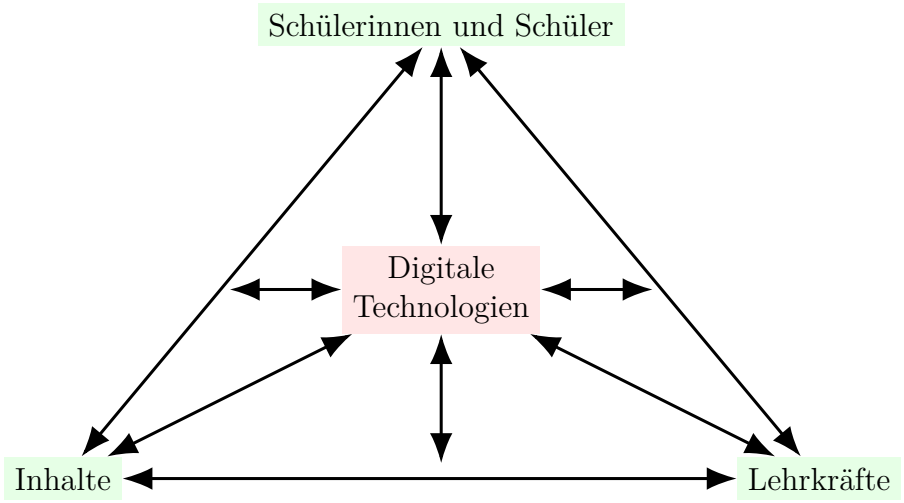


Abbildung 3.1.: Didaktisches Dreieck nach Schmidt-Thieme und Weigand (2015, S. 480)

stellung und Sichtweise [...] und wie drückt sich das aus“ (ebd., S. 480)? Diese Fragen stellen Schmidt-Thieme und Weigand (ebd.) zu dem Einsatz von DT im Unterricht. Aus Sicht der Lehrkräfte ist der Einsatz über veränderte Unterrichtsgestaltung und Methodik wichtig. Hinsichtlich des Inhalts ist zu prüfen, welche Themen durch DT in das Curriculum bewusst aufgenommen werden müssen und welche traditionellen Inhalte damit abgedeckt, verbessert oder verändert werden können. Somit sind, bezogen auf den Einsatz der digitalen Technologien, die Wechselwirkungen der Ecken des Dreiecks untereinander und mit der DT entscheidend (vgl. ebd., S. 480f). Daher stellt sich die Frage nach der „Gestaltung und [dem] adäquaten Design von Hard- und Software [...] [und] die Frage nach der Integration DT in den Unterricht“ (ebd., S. 481).

Eben dieser Frage versucht die Arbeit mit der Entwicklungsumgebung entgegenzutreten. Durch die Konzeption und Umsetzung eines neuen Werkzeugs sollen Inhalte anders vermittelt werden. Der mögliche integrierte Einsatz im Unterricht soll vor allem hinsichtlich der Wechselwirkungen zwischen den SuS und dem fachlichen Inhalt durch die Unterstützung der DT neue und veränderte Lernzugänge schaffen.

3.4. Repräsentationsformen im Mathematikunterricht

Ausgangspunkt der Repräsentationsformen im Mathematikunterricht bildet das in Abschnitt 3.3 angesprochene *E-I-S-Prinzip* durch die drei Darstellungsebenen *enaktiv*, *ikonisch* und *symbolisch*. Die Bedeutung des Darstellens wurde in jüngerer Zeit durch international verbreitete Standards für den Mathematikunterricht gestärkt, indem „das

Darstellen als typische Tätigkeit des Mathematik-Betreibens zu einer prozessbezogenen Kompetenz [angehoben wurde]“ (Jörissen und Schmidt-Thieme, 2015, S. 387). In den *Bildungsstandards im Fach Mathematik für den Mittleren Schulabschluss* sind unter anderem folgende Kompetenzen aufgeführt.

> **(K 4) Mathematische Darstellungen verwenden**
> Dazu gehört:
>
> – verschiedene Formen der Darstellung von mathematischen Objekten und Situationen anwenden, interpretieren und unterscheiden,
> – Beziehungen zwischen Darstellungsformen erkennen,
> – [...].
>
> **(K 5) Mit symbolischen, formalen und technischen Elementen der Mathematik umgehen**
> Dazu gehört:
>
> – [...]
> – symbolische und formale Sprache in natürliche Sprache übersetzen und umgekehrt, [...]
> – mathematische Werkzeuge (wie [...] Software) sinnvoll und verständig einsetzen.
>
> **(K 6) Kommunizieren**
> Dazu gehört:
>
> – Überlegungen, Lösungswege bzw. Ergebnisse dokumentieren, verständlich darstellen und präsentieren, auch unter Nutzung geeigneter Medien,
> – Die Fachsprache adressatengerecht verwenden,
> – Äußerungen von anderen und Texte zu mathematischen Inhalten verstehen und überprüfen.
>
> (KMK (2003, S. 8f))

Da die Arbeit im Rahmen der Grundschule stattfindet, ist nun zu prüfen welche Grundlagen hier vorausgesetzt werden. In den äquivalenten *Bildungsstandards Mathematik für den Primarbereich* sind Standards für allgemeine mathematische Kompetenzen aufgeführt. Unter anderem auch das Darstellen (vgl. KMK, 2004, S. 8). Um den konkreten Bezug zum Unterricht herzustellen, ist das Kerncurriculum (im vorliegenden Fall des Landes Niedersachsens) heranzuziehen. In diesem heißt es, „Mathematisches Arbeiten erfordert das Erstellen, die Auswahl und das Interpretieren von Darstellungen sowie das

Nutzen geeigneter Arbeitsmittel. Außerdem ist der flexible Wechsel von Darstellungen zwischen verschiedenen Repräsentationsebenen [...] unerlässlich“ (Niedersächsisches Kultusministerium, 2017, S. 8). Auch wenn Darstellen sogar einen separaten Bereich in den prozessbezogenen Kompetenzen erhält, so ist dieser (gemäß den Repräsentationsebenen) auch in anderen Kompetenzen zu finden. Als Beispiel sei hier das Kommunizieren zu nennen. „Kommunizieren im Mathematikunterricht beinhaltet die Fähigkeit, eigene Vorgehensweisen zu beschreiben und Lösungswege anderer zu verstehen. Die Schülerinnen und Schüler reflektieren gemeinsam über die Sache und über das eigene Denken“ (ebd., S. 7). Einen weiteren besonderen Stellenwert nimmt das Thema Medien im Kerncurriculum ein.

> „Im Mathematikunterricht werden vielfältige Medien eingesetzt, um mathematische Lernprozesse zu unterstützen. Diese sind mathematikdidaktisch reflektiert auszuwählen. Für digitale Medien gilt dies in einem besonderen Maße. Sie können allen Schülerinnen und Schülern einen Rahmen geben, um neue mathematische Handlungs- und Erfahrungsräume zu schaffen, Kommunikation und Interaktion zu unterstützen, Teilhabe zu ermöglichen sowie Eigenständigkeit und Wahrnehmung gezielt zu fördern.“ (ebd., S. 14)

Daraus lässt sich schließen, dass die Darstellungsebenen durch verschiedene Arten des Zugangs angesprochen werden. So sind Medien, genauso wie beispielsweise die natürliche Sprache, ein Zugang für die verschiedenen Darstellungsebenen, um die geforderten Kompetenzen zu fördern.

4. Ziele der Arbeit

Im Rahmen der Arbeit entsteht eine Android App. Diese Anwendung soll als Entwicklungsumgebung (IDE) für das Programmieren mit einer logischen Programmiersprache dienen.
Anwenderzielgruppe sind Kinder der 4. Klassenstufe. Da Programmiersprachen syntaktisch hohe Einstiegshürden (vgl. Modrow und Strecker, 2016, S. 25) für Kinder des entsprechenden Alters bürgen, ist ein Teilziel der Arbeit, die Entwicklungsumgebung so zu konzipieren und umzusetzen, dass sie die syntaktischen Schwierigkeiten herabsetzt. Dies soll mit Hilfe der natürlichen Sprache erreicht werden. Die Schülerinnen und Schüler sollen aktiv Fakten und Regeln formulieren, um sie einer Wissensbasis (Mini-Welt) (vgl. Abschnitt 2.2.4) hinzuzufügen. Des Weiteren soll es im Anschluss die Möglichkeit geben, Fragen natürlichsprachlich an das System zu stellen. Als Rückmeldung bietet die App eine visuelle Darstellung der derzeitigen Wissensbasis sowie akustische Antworten als Sprachausgabe auf gestellte Fragen. Hierbei soll sie auf die aktuellen Techniken und Frameworks (siehe Grundlagenkapitel 2) setzen und somit Wert auf eine konsistente und wartbare Architektur legen, um Erkenntnisse und technische Neuerungen im Nachhinein einfließen zu lassen.
Um einen Anwendungsfall realistisch darzulegen, muss eine Prüfung in einem didaktisch ausgearbeiteten Rahmen stattfinden. Hierfür bietet es sich an, Unterrichtsstunden zu einem Thema vorzubereiten. Diese sollten Teil des Kerncurriculums des entsprechenden Schulfaches sein. Somit können die Fragestellungen der Arbeit in einem qualitativ empirischen Rahmen untersucht und die Schülerinnen und Schüler beobachtet werden. Wichtig ist das Ansprechen der fachlichen, aber umso mehr der überfachlichen Kompetenzen.

Der Erfolg einer Forschungsarbeit hänge wesentlich vom Formulieren der Fragestellung(en) ab (vgl. Flick, 2011, S. 132f).
Es ergeben sich vier Leitfragen für die Arbeit. Diese sind sowohl im informatik-orientierten, als auch im didaktischen Bereich zu verorten.

1. Ist es möglich, in einer App die Intention durch natürlichsprachliche Eingaben von Kindern der 4. Klasse zu einem fachlichen Thema korrekt zu interpretieren?
2. Kann eine App mit einer HMI ausgestattet werden, die Kindern der 4. Klasse die Interaktion mit einem Prolog-Interpreter ohne syntaktische Hürden ermöglicht?
3. Wie kann die App intern strukturiert sein, um zukünftige Fortschritte bei Frameworks und Technologien zur einfachen Benutzerinteraktion ohne hohe Änderungsaufwände in der App nutzen zu können?
4. Hilft die App mit logischem Programmierparadigma und im erarbeiteten Unterrichtsrahmen Kindern der 4. Klasse, fachliche Zusammenhänge sprachlich zu formulieren? Werden die gewünschten Kompetenzbereiche angesprochen?

5. SLIDE - Speech and Logic IDE

In diesem Kapitel ist die entwickelte Anwendung beschrieben, die Gegenstand zur Beantwortung der gestellten Fragen ist. Der Einstieg befasst sich mit dem Definieren einer Product Vision, die Einordnung in einen Systemkontext und mit dem Aufstellen der Anforderungen als User Stories. Daraus entwickelt sich eine Architekturvorstellung für die App. Die Umsetzung der Architektur bildet das Grundgerüst, um hinsichtlich der Forschungsfrage eine anpassbare App zu entwickeln.

5.1. Product Vision

Eine, unter anderem in Scrum eingesetzte, und nach Foegen et al. (vgl. 2014, S. 214ff) hilfreiche Technik zur Beschreibung eines Produkts oder Projekts heißt *Product Vision*. Sie hat das Ziel als Leuchtturm für alle Projektbeteiligten zu fungieren. Dafür soll ein Satz genügen. Sie ist außerdem ein Einstiegspunkt zum Definieren von Anforderungen sowie ein Kontrollwerkzeug während des Projektverlaufs. Ich habe bisher immer gute Erfahrungen damit gemacht, eine Tabelle mit *Stakeholder*, *Zielgruppen*, *(Nutzer-)Bedürfnissen*, *Produkt* und *Wert* anzulegen. Ein Stakeholder ist „eine Person oder Organisation, die (direkt oder indirekt) Einfluss auf die Anforderungen des betrachteten Systems hat“ (Pohl und Rupp, 2011, S. 12). Das tabellarische Vorgehen hat mehrere Vorteile. Im Anforderungsmanagement ist es ohnehin nötig, alle relevanten Stakeholder zu identifizieren (vgl. ebd., S. 29ff). In dieser Form bietet es darüber hinaus eine Möglichkeit, vom Grobgranularen zum Feingranularen zu arbeiten. Konkrete Zielgruppen bilden hierbei immer eine Untermenge der Stakeholder. Die anschließenden Bedürfnisse, Produkte und Werte können den jeweiligen Zielgruppen zugeordnet werden (siehe Product Vision im Anhang A.3). Interessant sind auch Überschneidungen und Unterschiede. In der erstellten Product Vision sind die farbigen Zuordnungen gewählt, wenn vorrangig *eine* Zielgruppe das Produkt, Bedürfnis oder den Wert besonders stark vertritt. Die ungefärbten Produkt- oder Featurebeschreibungen gelten für mehrere Zielgruppen. Ebenfalls kann

nun eine Priorisierung der Zielgruppen vorgenommen, sowie eventuelle Barrieren/Risiken hinzugefügt werden. Eine Priorisierung ist in diesem Fall nicht nötig. Eine mögliche Barriere stellt das Nichtvorhandensein von Smartphones/Tablets dar, falls die Schule nicht über solch eine Ausstattung verfügt und die Hardware leihen müsste. Ein weiteres potentielles Risiko bei Sprachein- und ausgabe ist der Lärmpegel im Klassenunterricht. Solche Barrieren sollten während des gesamten Projektverlaufs kontinuierlich bedacht werden. Aus den erarbeiteten Punkten aus der Tabelle A.3 im Anhang lässt sich folgende Product Vision formulieren.

> Wir benötigen eine durch Sprache bedienbare Programmierumgebung als App, die einfach in den bestehenden Unterricht für unerfahrene LehrerInnen und SchülerInnen integriert werden kann, um fachliche und überfachliche Kompetenzen zu fördern.

In die Product Vision-Tabelle wird neben den Schülerinnen und Schülern sowie den Lehrpersonen auch der Entwickler als Zielgruppe aufgenommen. Die Bedürfnisse dieser Zielgruppe müssen im Rahmen dieser Arbeit ebenfalls bedacht werden. Somit kann die Product Vision noch um den Nebensatz der einfachen Anpassbarkeit erweitert werden, oder um der Lesbarkeit entgegen zu kommen, als zweiten Satz angefügt werden.

> **Version 1**
> Wir benötigen eine durch Sprache bedienbare Programmierumgebung als App, die einfach in den bestehenden Unterricht für unerfahrene LehrerInnen und SchülerInnen integriert werden kann, um fachliche und überfachliche Kompetenzen zu fördern und die durch Software- bzw. Architekturentscheidungen ohne hohe Änderungsaufwände an die sich verändernden, technischen Rahmenbedingungen angepasst werden kann.

> **Version 2**
> Wir benötigen eine durch Sprache bedienbare Programmierumgebung als App, die einfach in den bestehenden Unterricht für unerfahrene LehrerInnen und SchülerInnen integriert werden kann, um fachliche- und überfachliche Kompetenzen zu fördern.
>
> Durch Software- bzw. Architekturentscheidungen soll erreicht werden, dass sich verändernde, technische Rahmenbedingungen ohne hohe Änderungsaufwände angepasst werden können.

Somit deckt sich die Product Vision inhaltlich mit den Zielen in Kapitel 4.

5.2. Systemkontext

Der Systemkontext ist nach Pohl und Rupp (2011) „der Teil der Umgebung eines Systems, der für die Definition und das Verständnis der Anforderungen des betrachteten Systems relevant ist“ (ebd., S. 21). Aspekte wiederum, die in einem Systemkontext betrachtet werden können, sind Personen, andere technische Systeme beziehungsweise Hardware, Geschäftsprozesse, Ereignisse und Dokumente (vgl. ebd., S. 22). Ein Anwendungsfalldiagramm (Use-Case-Diagram) wird oftmals zur Dokumentation und Analyse des Systemkontextes verwendet (vgl. Jacobson, 1992, S. 127ff und S. 153ff).

Im vorliegenden Fall sind zwei Systeme modelliert. Zum einen handelt es sich um die Android App, mit der die Lernenden sowie die Lehrperson interagieren. Zum anderen befindet sich darunter das in Dialogflow umgesetzte Backend, das eine Schnittstelle zur App bereitstellt. Es existiert ein zentraler Anwendungsfall. Dem Nutzer bzw. der Nutzerin (den Lernenden) ist es möglich eine *Wissensbasis zu erstellen*. Als Erweiterung ist es sinnvoll, drei weitere Anwendungsfälle zu diesem hinzuzufügen. Die Lernenden sollen ein Expertensystem (siehe Abschnitt 2.2.5) programmieren. Die drei Anwendungsfälle (*Regel erstellen*, *Fakt erstellen* und *Wissensbasis anfragen*) beziehen sich auf das Expertensystem und auf die entsprechende Sprache (siehe Abschnitt 2.2.4) Sie sind Teil der Wissensbasis oder fragen diese an. *Wissensbasis einsehen* ist neben den Lernenden für die Lehrperson von Bedeutung. Durch die grafische Darstellung der derzeitigen Wissensbasis lässt sich das derzeitige Wissen erkennen und überprüfen.
Die drei Anwendungsfälle *Regel erstellen*, *Fakt erstellen* und *Wissensbasis anfragen* können nur ausgeführt werden, wenn die Intention und die Entitäten durch das Backend ermittelt wurden. Ein Backend (oder ein im Hintergrund dafür vorgesehenes System) wird benötigt, da die Kommunikation mit der Anwendung durch natürliche Sprache realisiert werden soll (siehe Abschnitt 2.5). Daher dient die Android App als Aktor gegenüber diesem Dienst. Die Entscheidung, das Backend und die App als voneinander getrennte Systeme zu betrachten, hat folgenden Hintergrund. Die Backendlogik könnte bei einem Technologiewechsel ausgetauscht oder gar in die App integriert werden. Da diese Logik aber im hier vorliegenden Fall ein separates System übernimmt, das auch für sich stehend verwendet werden kann, ist die Aufteilung und Abgrenzung gerechtfertigt. Somit sind beide Systeme, die Aktoren und die Systemkontextgrenze (der Bereich außerhalb des Diagramms) definiert und voneinander abgegrenzt (siehe Abbildung 5.1).

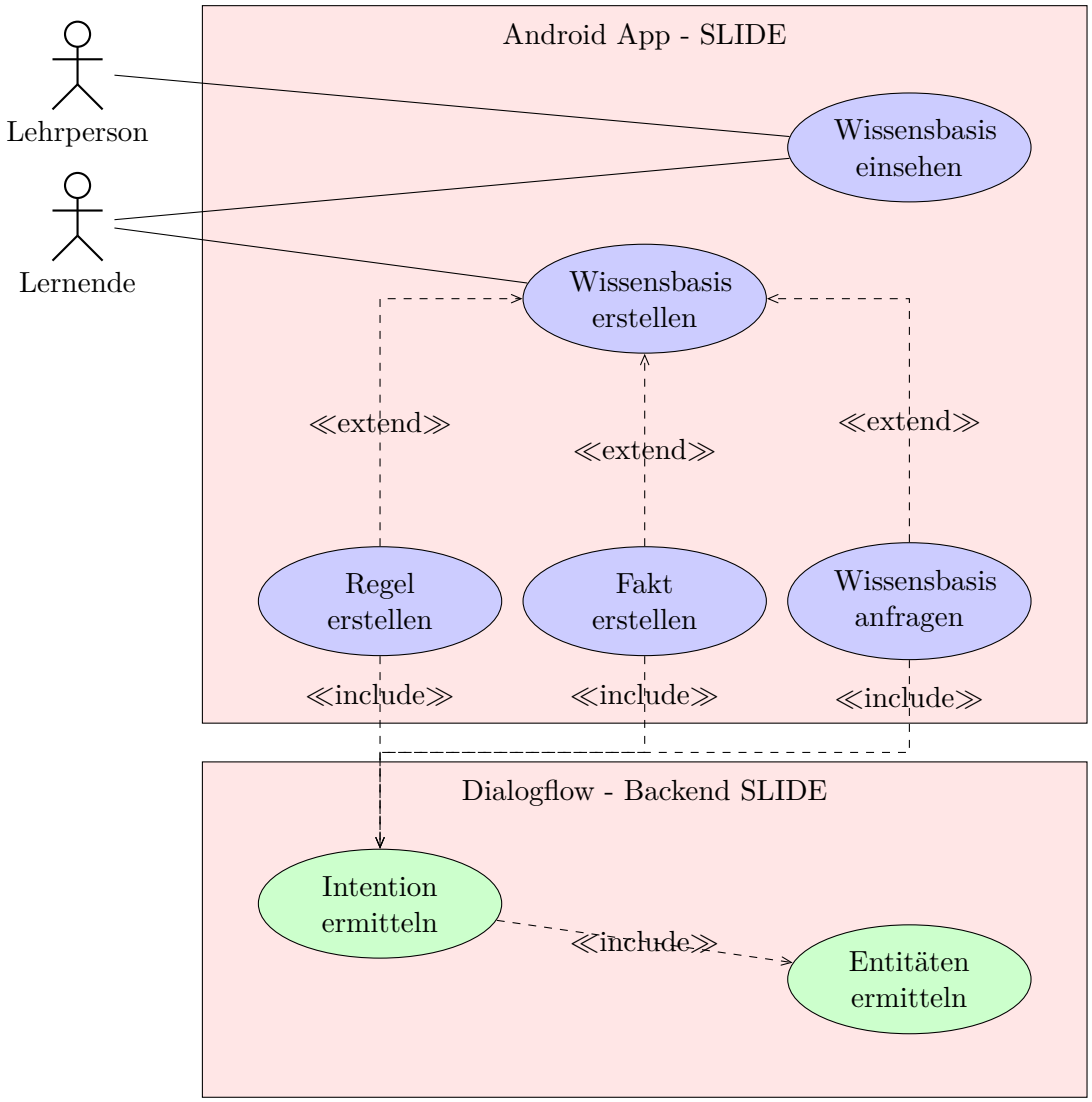


Abbildung 5.1.: Use-Case-Diagram zur Android Anwendung

5.3. Anforderungen

Gemäß der Anwendungsfälle aus Abschnitt 5.2 lassen sich gemeinsam mit den Zielen der Arbeit und der Product Vision die Anforderungen aufstellen, die im Softwareentwicklungsprozess umgesetzt werden. Um den Entwicklungsprozess so schlank wie möglich zu halten, sind die Anforderungen in *Epics* und *User Stories* beschrieben. „Eine User Story besteht typischerweise nur aus einem oder ein paar wenigen Sätzen und beschreibt einen Anwendungsfall auf grober Ebene“ (Preußig, 2015, S. 93). Epics wiederum fassen mehrere User Stories zu einer zusammen. Das habe den Vorteil, mehrere Anwendungsfälle zu Teilbereichen zuordnen zu können. Dies sei vor allem zu Beginn eines Projektes vorteilhaft (vgl. ebd., S. 94). Eine User Story beschreibt einen konkreten, sichtbaren

Mehrwert für den Kunden (vgl. Wirdemann und Mainusch, 2017, S. 50) und somit auch für die Nutzer. Wirdemann und Mainusch sprechen vom erhöhten *Geschäftswert* durch das Umsetzen der entsprechenden User Story. Des Weiteren sei ein häufiges Missverständnis, dass die Story für sich stehend funktioniere. Hierbei würden aber die beiden zentralen Werkzeuge beim Verwenden der User Story unterschlagen. Die Story stehe lediglich für eine Kommunikationsgrundlage und funktioniere nur mit entsprechenden Akzeptanzkriterien, die im Laufe des Projektes angepasst werden. Trotz des scheinbar geringen Informationsgehalts, der in einem Satz enthalten sein kann, ist das Beschreiben der User Story von großer Wichtigkeit. Ein Satzmuster für eine Story-Karte hat sich bewährt.

> Als <Benutzerrolle> will ich <das Ziel>[, so dass <Grund für das Ziel>].
> (ebd., S. 57)

Der Grund kann hierbei auch mit *um zu* beschrieben werden. Wichtig ist lediglich, dass der Geschäftswert bereits in der Formulierung der Story bewusst bedacht wird. Dieses Muster wurde ursprünglich von Mike Cohn entworfen und hat sich als passende, einfache Beschreibung durchgesetzt (vgl. ebd., S. 57). Epics wiederum können ebenfalls so beschrieben werden, „müssen [aber] nicht zwingend dem User Story-Muster [...] folgen“ (ebd., S. 59). Im Anhang A.4 sind die entsprechenden Epics und User Stories aufgelistet. Auf einige werde ich im Folgenden näher eingehen.
Hinsichtlich der Anwendungsfälle aus Abschnitt 5.2 lassen sich Epics definieren. Diese sind im vorliegenden Fall bereits mit der App in Verbindung gebracht. Somit heißt ein Epic *Mini-Welten*. Es enthält User Stories, die sich mit dem Erstellen und Verwalten der einzelnen Wissensbasen beschäftigen. Das Epic *Wissen aufnehmen* umfasst die Anwendungsfälle *Regel erstellen*, *Fakt erstellen* und *Wissensbasis einsehen*. *Wissensbasis anfragen* wiederum wird durch die User Stories des Epics *Wissen erfragen* abgedeckt. Die Funktionalität im Backend wird durch das Epic *Intentionen verstehen* zusammengefasst.

5.3.1. User Story

Eine Story solle unabhängig vom *WIE* das *WAS* beschreiben. „User Stories sind frei von technischem Jargon [und] [...] treffen keinerlei Annahmen über die Benutzeroberfläche. Eine Story beschreibt ausschließlich das Ziel einer Rolle [...]“ (ebd., S. 61). Cohn (2004) stellt außerdem sechs Eigenschaften zusammen, die eine gute User Story ausmachen sol-

len. Die sechs Eigenschaften können mit dem Akronym *INVEST* abgekürzt und gemerkt werden.

1.	I	Independent
2.	N	Negotiable
3.	V	Valuable
4.	E	Estimatable
5.	S	Small
6.	T	Testable

Tabelle 5.1.: INVEST - Eigenschaften guter User Stories nach Cohn (2004, S. 17ff)

Wie eingangs in Abschnitt 5.3 beschrieben, reicht die reine Beschreibung der User Story nicht aus. Es müssen Akzeptanzkriterien angefügt werden, um die Story sinnvoll implementieren zu können. Hier kann nun das *WIE* genauer ausdefiniert werden, was beispielsweise auch Oberflächen und Designaspekte betrifft. Die User Stories in Tabelle A.4 sind *vertikal* geschnitten. Dies bedeutet, dass die daraus resultierenden Aufgaben die gesamte Architektur betreffen. Die Implementierungsaufgaben sind in allen Schichten der Anwendung durchzuführen. Ich werde dies am Beispiel der Akzeptanzkriterien und Aufgaben der User Story 18 *Als Lernender möchte ich eine Mini-Welt für Fakten und Regeln erstellen, um für jeden Anwendungsfall eine Umgebung zu haben.* darstellen. Folgende Punkte können Teil der Akzeptanzkriterien sein.

1. Klickaktion zum Erstellen einer Mini-Welt
2. Android Dialog öffnet sich
3. Inputfeld für einen Namen
4. Erstellen muss mit *Speichern* bestätigt werden
5. Abbrechen ist mit *Abbrechen* möglich
6. Der Nutzer erhält durch den Text *Gib deiner Welt einen Namen.* einen Hinweis für das Eingabefeld
7. Erstellen geht nur, wenn der Name...
   a) ... mindestens ein valides Zeichen enthält
   b) ... der Name noch nicht vergeben ist
   c) ... nur aus validen Zeichen besteht ([A-Za-zäÄüÜöÖß0-9\s]*)
8. Der Benutzer erhält spezifisches Feedback, wenn ein Fehler aufgetreten ist
   a) Dein gewählter Name muss mehr als ein Zeichen enthalten
   b) Dein gewählter Name existiert bereits
   c) Dein gewählter Name darf nur aus Zahlen, Buchstaben und Leerzeichen bestehen

9. Aktionen, Benutzerflow, Error-Handling und Design müssen den Google Design Guidelines entsprechen[1]

Alternativ können weitere Dokumente wie Abläufe in Form von (Sequenz-)Diagrammen oder Screendesigns der Designer als Akzeptanzkriterien mit angefügt werden. Darauf wurde in diesem Fall verzichtet.
Aus der User Story mit den angefügten Akzeptanzkriterien lassen sich mehrere Aufgaben definieren, die dann im Entwicklungszyklus umgesetzt werden.

- Benötigte Texte anlegen und internationalisieren (4, 5, 6, 8 a-c)
- Button zum Hinzufügen einer neuen Mini-Welt (Guidelines 5.3.2)
- Präsentationsschicht implementieren (vgl. Abschnitt 5.4)
  - ViewModel erstellen
  - Presenter implementieren
  - View implementieren
  - Contract implementieren
  - Inputvalidierung erstellen
- Domänenlogik implementieren (vgl. Abschnitt 5.4)
  - Unit-Tests erstellen
  - Model erstellen
  - UseCase *AddMiniWorldProfileUseCase* erstellen
- Datenschicht implementieren (vgl. Abschnitt 5.4)
  - DataModel erstellen
  - Abfrage, ob der Name vergeben ist
  - Serialisieren des Objektes
  - Speichern in die Datenbank

Dies ist lediglich ein Ausschnitt möglicher konkreter Aufgaben, der eine Idee des Vorgehens vermitteln soll. Im größeren Projektumfeld würden dort mindestens noch die Aufgaben hinzukommen, die nach der *Definition of Done* wesentlich sind. Die Definition of Done (DoD) ist eine Vereinbarung, in der projekt- und teamspezifische Anforderungen an eine User Story stehen. Sie definiert den Status *fertig*. Kriterien könnten beispielsweise die folgenden sein.

- Alle Akzeptanzkriterien wurden durch manuelle und/oder automatisierte Tests abgedeckt und sind bestanden
- Der Pull-Request zur User Story wurde von mindestens einem anderen Entwickler geprüft

[1] https://material.io/guidelines/

- Alle Aufgaben haben den Status *fertig*

5.3.2. Design

Dieser Abschnitt beschäftigt sich mit den optischen Designentscheidungen in der App. Zum einen wird die Darstellung der Fakten und Regeln in der Wissensbasis dargelegt und zum anderen das Material-Design von Google vorgestellt. An Beispielen wird gezeigt, welche Einflüsse das Material-Design bei der Umsetzung der App hat.

Darstellung der Regeln und Fakten

Die Interaktion mit der App geschieht zu großen Teilen mit der natürlichen Sprache. Das bedeutet, dass das Hinzufügen von Regeln und Fakten sowie das Anfragen der Wissensbasis durch Antippen des Mikrofon-Buttons und Sprechen in das Gerät geschieht. Ein Feedback beim Erstellen von Fakten und Regeln ist sprachlich nicht ideal umsetzbar. Um den derzeitigen Stand der Wissensbasis zu visualisieren, bietet sich das Display des Smartphones bzw. Tablets an. Für die Darstellung des Wissensstands wird eine Syntax geschaffen, wofür alle Regeln und Fakten an einem fiktiven Seil befestigt werden. Diese Analogie zum materiellen Seil soll den Lernenden helfen, das Modell einer Wissensbasis zu verstehen. Jeder Bestandteil des Seils ist Teil der Mini-Welt und somit relevant beim Auswerten der Wissensbasis. Die Fakten und Regeln werden als Karten dargestellt, die an das Seil geknüpft sind. Regeln werden orange und Fakten gelb eingefärbt. Da Regeln aus einem Regelkopf und Bedingungen bestehen und diese Bedingungen wiederum Fakten sind, ergibt sich eine Kette aus Karten. Abbildung 5.2 zeigt die Darstellung und damit die Syntax der Wissensbasis.

Android Material Design Guidelines

Google, das Unternehmen, das die Android-Plattform zur Verfügung stellt, gibt Entwicklern das Material Design an die Hand, um Software zu designen. Dabei ist das Material Design nicht lediglich auf Android Apps limitiert. So sind unter *Platforms* neben Android auch Web und Apples iOS zu finden. „Material Design supports design and usability best practices across platforms to help create beautiful user experiences“ (Google, 2017d). Im Folgenden wird an drei Beispielen dargestellt, wie diese Guidelines in der Anwendung wiederzufinden sind. Das erste Beispiel beschäftigt sich mit der Darstellung

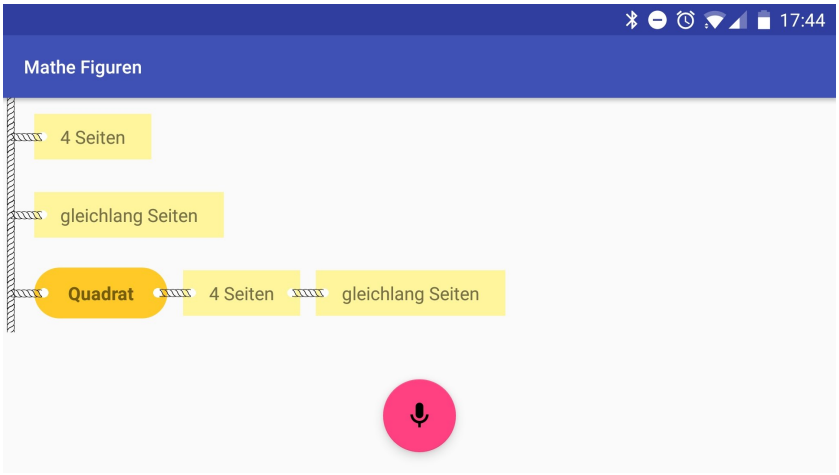


Abbildung 5.2.: Syntax der Wissensbasis mit Regeln und Fakten

der Mini-Welten (siehe Abbildung 5.3).

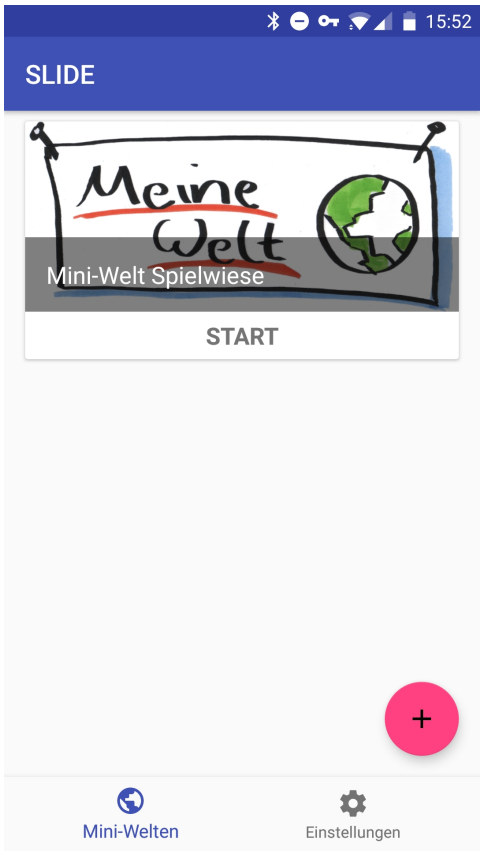


Abbildung 5.3.: App Einstieg SLIDE

Die User Story mit der Id 8 beschreibt: *Als Lernender möchte ich eine Übersicht über alle Mini-Welten haben, um eine gespeicherte auswählen zu können.* Die Übersicht wurde in einer Liste aus *Karten* realisiert. Die Design Guidelines besagen „Cards are a convenient means of displaying content composed of different elements. They're also well-suited for showcasing elements whose size or supported actions vary, like photos with captions of variable length“ (Google, 2017b). Solch eine Anforderung ist hier umgesetzt. Die verschiedenen Elemente sind ein passendes (derzeit statisches) Bild, ein Textlabel mit dem Namen der Mini-Welt sowie ein Button, der das Starten in die Mini-Welt aktiviert. Somit bieten die Karten verschiedene Mini-Welten an, auf die zugegriffen werden kann.

Das zweite Beispiel ist der *Floating Action Button*. „A floating action button represents the primary action in an application“ (Google, 2017a). Dies ist zum einen das Erstellen neuer Mini-Welten (siehe Abbildung 5.3) in der Einstiegs-View. Zum anderen ist die Hauptaktion innerhalb einer Mini-Welt die

Spracheingabe, um Fakten bzw. Regeln zu erstellen und die Wissensbasis anzufragen (siehe Abbildung 5.4). Dies entspricht auch den im Material Design angesprochenen Qualitätskriterien. So sollen Floating Action Buttons lediglich für positive Aktionen wie *Create*, *Share* oder *Activate Microphone* genutzt werden, nicht aber als Löschaktionen oder limitierte Aktionen wie Ausschneiden (Google, 2017a, vgl.). Somit ist der Floating Action Button das zentrale Element in der Wissensbasis und soll einen möglichst direkten und intuitiven Zugang zur Sprachsteuerung ermöglichen.

Das dritte Beispiel verwendet die Entwurfsmuster zur Texteingabe und deren Fehlerbe-

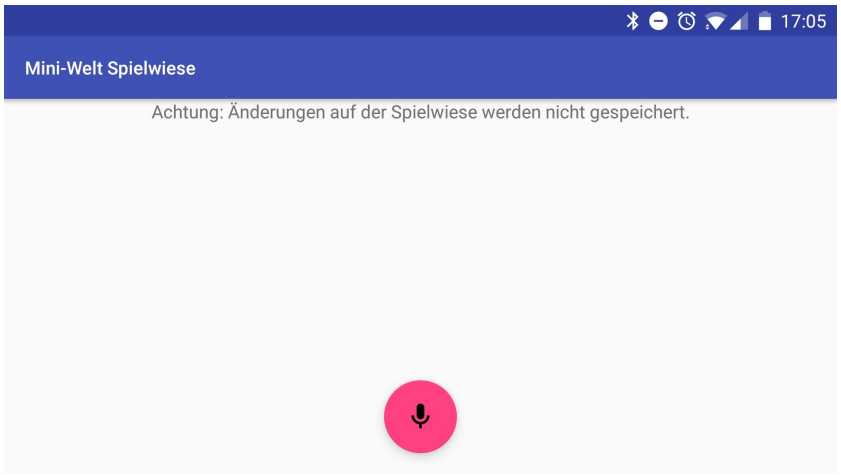


Abbildung 5.4.: Floating Action Button Microphone

handlung. Zu *Text field input* gibt die Guideline vor, dass Hilfetexte vor, während oder nach einer Interaktion des Nutzers mit dem entsprechenden Feld hinzugefügt werden sollen. Fehlertexte sollten nur nach der Eingabe inkorrekter Daten erfolgen (Google, 2017c, vgl.). Dies wurde im Dialog zum Erstellen einer neuen Mini-Welt umgesetzt (User Story 18 *Als Lernender möchte ich eine Mini-Welt für Fakten und Regeln erstellen, um für jeden Anwendungsfall eine Umgebung zu haben.* - siehe auch Abschnitt 5.3.1). Abbildung 5.5 zeigt das Eingabefeld mit und ohne Fehlermeldung. Der Fehlerfall tritt auf, sobald der Nutzer das Speichern bestätigt, aber keinen Namen für die Mini-Welt vergeben hat. Diese direkte und klare Rückmeldung soll dem Nutzer helfen, die passenden Eingaben zu tätigen.

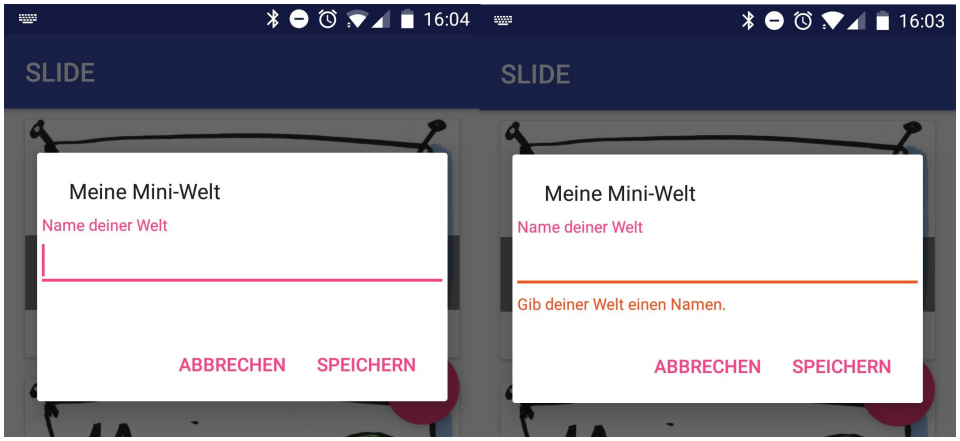


Abbildung 5.5.: Fehlerbehandlung im Eingabefeld

5.4. Architektur

Um dem Ziel *Wie kann die App intern strukturiert sein, um zukünftige Fortschritte bei Frameworks und Technologien zur einfachen Benutzerinteraktion ohne hohe Änderungsaufwände in der App nutzen zu können?* (vgl. Kapitel 4) gerecht zu werden, soll die Android App nach den Design Prinzipien und dem Clean Architecture Ansatz umgesetzt werden. Da die Architektur von Martin (2018) frameworkunabhängig sei (vgl. Abschnitt 2.3.4), sollte es möglich sein, im Rahmen des Android-Frameworks diese Architektur zu implementieren. Das Single Responsibilty Principle und das Open-Closed Principle (siehe Abschnitt 2.3.1) finden in den ersten Überlegungen vor allem auf Modulebene statt. Damit ergibt sich eine mögliche Aufteilung in drei Module.

- **app** - das Standardmodul für eine Android Anwendung. Hier soll vor allem die UI-Schicht implementiert sein, da die spezifischen *Activity* und *Fragment* Android-Klassen den Lifecycle über die Views bestimmen.

- **data** - dieses Modul beinhaltet Funktionalitäten, die Abhängigkeiten zu externen Quellen besitzen. Dazu zählen ein Service für einen Prolog Interpreter, die Datenbank (SQLite), die Sprachverarbeitung sowie die Schnittstelle zum Dialogflow SDK, um Webschnittstellenaufrufe durchzuführen.

- **domain** - das Domain-Modul bildet den Kern der Anwendung und enthält die Businesslogik.

Somit implementieren *app* und *data* die Details aus, wohingegen das Domain-Modul abstrakte Logik enthält. Da auch Android ein Implementierungsdetail ist, kann (und muss)

das Domain-Modul frei von diesem Framework aufgebaut sein. In der Entwicklungsumgebung ist dies als reines Java/Kotlin-Modul deklariert. Somit können keine android-spezifischen Aufrufe stattfinden. Abbildung 5.6 zeigt die drei Module. Das Domain-

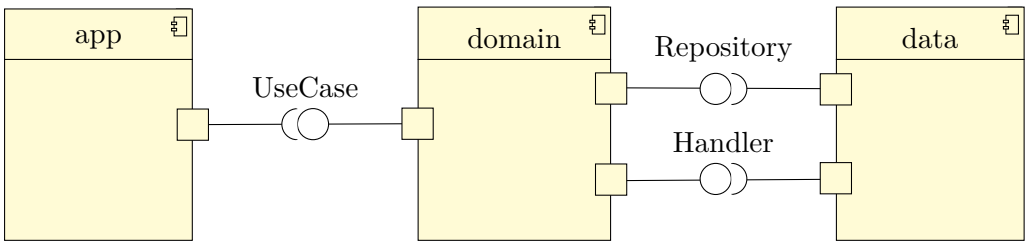


Abbildung 5.6.: SLIDE - Module

Modul bietet durch die UseCases Observable-Objekte an, auf die sich das App-Modul registrieren kann (siehe Abschnitt 5.4.4). Die Kommunikation zwischen dem Data-Modul und dem Domain-Modul geschieht durch das Implementieren des Repository-Patterns (siehe Abschnitt 2.3.2). Das Domain-Modul ruft die Schnittstellen auf, die durch das Data-Modul implementiert werden. Durch diese Möglichkeiten der *Output-Ports* kann die Kommunikation mit den anderen Modulen gelingen, ohne Abhängigkeiten in deren Richtung zu haben. Die Kommunikation mit einem ServiceHandler im Data-Modul (siehe Abschnitt 5.4.3) ist nach demselben Prinzip (Dependency Inversion Principle siehe Abschnitt 2.3.1) implementiert wie das Repository-Pattern.

5.4.1. App-Modul

Das App-Modul ist hauptsächlich für das UI zuständig. Dies impliziert im Android-Kontext die jeweiligen Activities und Fragments. Fragments sind wie Activities View-Komponenten mit eigenem Lebenszyklus. Dieser unterscheidet sich zwar in einigen Aspekten von denen einer Activity, ist aber grundlegend ähnlich. Die Unterschiede haben keine Relevanz für das Verständnis. Implementiert ist ein *Model-View-Presenter* (MVP) bzw. *(Model)-View-Presenter* Pattern. Die Abbildung 2.10 im Abschnitt 2.3.3 sollte für das Verständnis der Zusammenhänge herangezogen werden. Der dargestellte Aufbau und die Beziehungen wurden im Folgenden an konkreten Klassen vollzogen.

Model - ViewModel

Auf das Model im Sinne des MVP (siehe Abschnitt 2.3.3) hat der Presenter in diesem Fall keinen direkten Einfluss. Er interagiert über das Domain-Modul mit der Business-

logik. Spezifische Model-Objekte in diesem Modul gibt es jedoch trotzdem. Diese heißen ViewModel-Objekte und sind in passender Datenstruktur für die View vorhanden. Ein Objekt-Mapper mappt die Model-Objekte aus dem Domain-Modul zu ViewModel-Objekten im App-Modul und umgekehrt. Wieso ist solch ein Mapping nötig? An zwei ViewModel Beispielen wird die Relevanz verdeutlicht.

Beispiel 1: Parcelable-Interface

Das Datenobjekt einer Mini-Welt und deren Eigenschaften müssen durch einen Android-*Intent* an die nächste Activity oder in das nächste Fragment weitergegeben werden. Hierfür wird das Objekt serialisiert in ein *Bundle* verpackt (siehe Abschnitt 2.1.1). Diese Serialisierung über das Interface *Parcelable* ist android-spezifisch. Demnach ist es ein Implementierungsdetail und nicht relevant für das Domain-Modul. Das ViewModel-Objekt (siehe Anhang B.3) muss somit separat im App-Modul implementiert und an dessen Anforderungen angepasst werden. Die Entscheidung, das Domain-Modul als reines Java/Kotlin-Modul zu klassifizieren, hält darüber hinaus den Entwickler hier ebenfalls sinnvollerweise davon ab, das Parcelable-Interface durch das Model-Objekt implementieren zu lassen. Auf das im Package *android.os* verortete Interface kann durch die fehlende Android-Abhängigkeit im Domain-Modul nicht zugegriffen werden.

Beispiel 2: Android-View-Klasse

Ein zweites Beispiel ist eine konkrete Android-View-Klasse, von der geerbt werden kann. Eine Activity kann dieses ViewModel direkt im Layout darstellen. Quelltext 5.1 zeigt das bedingte Umwandeln in die konkreten ViewModel-Objekte.

Quelltext 5.1.: Mapping von Model zu ViewModel-Objekten in KnowledgeViewMapper.kt

```
override fun transformMtoVM(model: LogicItem?):
 ↪ KnowledgeView? {
 when (model) {
  is SingleFact -> return SingleFactView(context, model)
  is PropertyFact -> return PropertyFactView(context, model)
  is CommonRule -> return CommonRuleView(context, model)
 }
 return null
}
```

Das View-Objekt kennt spezifische Logik zur Darstellung des jeweiligen ViewModels selbst (vgl. init-Methode Anhang B.4 Zeile 15-18 mit Anhang B.5 Zeile 13-29), sodass auch dieses *View*-Model im erweiterten Verständnis Logik enthält. Diese ist aber zur

Businesslogik abzugrenzen, die im Domain-Modul zu implementieren ist. Sie beschränkt sich lediglich auf darstellende Aspekte.

View - Activity

Wie im vorangegangenen Abschnitt beschrieben erhält die Activity-Klasse ViewModel-Objekte, die dann als Layout-Objekte auf der Oberfläche angezeigt werden. Des Weiteren implementiert die Activity-Klasse einen Contract (siehe Abschnitt 2.3.3 und Quelltext 5.2), der als Kommunikationsschnittstelle von und zum Presenter fungiert.
Exemplarisch kann die *IDEActivity* herangezogen werden. Diese Activity der Android App implementiert alle Funktionen des *IDEContract.View-Interfaces* (siehe Quelltext 5.2 Zeile 2).

Quelltext 5.2.: IDEContract.kt

```
1  interface IDEContract {
2   interface View : BaseView {
3    var ideWorkspaceViewModel: IDEWorkspaceViewModel?
4    fun setIDEViewModel(ideWorkspaceViewModel:
       ↪ IDEWorkspaceViewModel?)
5    fun getProfileModel(): MiniWorldProfileViewModel?
6    fun getSelectedKnowledge(): List<KnowledgeView>?
7    fun setKnowledgeUnSelected()
8   }
9   interface Presenter : BasePresenter<View> {
10   fun performUserInput()
11   fun deleteKnowledge()
12  }
13 }
```

Der Presenter (siehe Abschnitt 5.4.1) hat durch den Contract ebenfalls Zugriff auf diese Funktionen. Er greift über die Referenz des View-Interfaces auf die Activity-Interface-Methoden zu (vgl. The Dependency Inversion Principle Abschnitt 2.3.1). So erhält die IDEActivity Daten, die bereits aus ViewModel-Objekten bestehen (siehe Quelltext 5.3 und vgl. Quelltext 5.2 Zeile 4).

Quelltext 5.3.: IDEPresenter - Mapping und Aufruf der View-Methode

```
override fun onNext(t: IDEWorkspaceModel?) {
 super.onNext(t)
 view?.setIDEViewModel(ideWorkspaceVMMapper
  ↪  .transformMtoVM(t))
}
```

Bei Änderungen oder Aktionen auf der View ruft die *IDEActivity* entsprechende Presenter-Methoden auf (siehe Quelltext 5.4 Zeile 2 und vgl. 5.2 Zeile 10).

Quelltext 5.4.: IDEActivity - onRequestPermissionsResult

```
1 if (...){
2  idePresenter.performUserInput()
3 }else{
4  error(getString(R.string.error_need_permission))
5 }
```

Die Referenz auf den *IDEPresenter* erhält die Activity durch *Dependency Injection*, wobei dessen Instanz in ein Feld injiziert wird. Dependency Injection ist der Aufbau eines azyklischen Graphen mit Objektinstanzen und deren Abhängigkeiten. In Folge dessen können die daraus entstandenen Instanzen in beliebige Objekte entlang des Graphen injiziert werden.

Presenter

Der Presenter bildet zum einen die Schnittstelle zum Domain-Modul und zum anderen steuert er die Daten- und Aktionskommunikation mit der Präsentationsschicht (View). Im Quellcodeausschnitt 5.5 ist der Konstruktor einer Presenter-Klasse zu sehen. Die Referenzen auf die UseCases, um mit der Domain-Ebene zu interagieren, und die Datenmodell-Mapper werden über Dependency Injection injiziert. Der *IDEPresenter* implementiert ein Basis-Presenter-Interface, das einen generischen Typen als Feld beinhaltet (siehe Quelltext 5.6 Zeile 2). In diesem Fall (siehe Quelltext 5.5 Zeile 10) ist es die Referenz auf das *IDEContract.View-Interface*. Somit erhält der Presenter Zugriff auf die im Contract definierten Methoden, die durch die IDEActivity-View implementiert sind.

Quelltext 5.5.: IDEPresenter - Konstruktor

```
class IDEPresenter @Inject constructor(
 /* Use Cases the presenter needs to execute */
 private val executeUserRequestUseCase:
  ↪ ExecuteUserRequestUseCase,
 private val getIDEWorkspaceUseCase: GetIDEWorkspaceUseCase,
 private val deleteKnowledgeUseCase: DeleteKnowledgeUseCase,
 /* Mapper: Model -> ViewModel and ViewModel -> Model */
 private val miniWorldProfileVMMapper:
  ↪ MiniWorldProfileVMMapper,
 private val ideWorkspaceVMMapper: IDEWorkspaceVMMapper,
 private val knowledgeViewMapper: KnowledgeViewMapper
) : BasePresenter<IDEContract.View>, IDEContract.Presenter {
```

Quelltext 5.6.: BasePresenter.kt

```
interface BasePresenter<V : BaseView> {
 var view : V?
 fun resume()
 fun pause()
 fun destroy()
}
```

5.4.2. Domain-Modul

Das Domain-Modul ist der Kern der Anwendung. Es besteht zum Großteil aus den Use-Cases und den Entitäten (Model-Objekte). Durch das *Repository*-Pattern (siehe Abschnitt 2.3.2) wird der *Output-Port* für das Data-Modul bereitgestellt. Das bedeutet, dass alle Aktionen Richtung externer Schnittstellen über solch ein Interface von einer UseCase-Klasse aufgerufen werden. Quellcode 5.7 zeigt einen Ausschnitt des *IDERepository*-Interface.

Quelltext 5.7.: Ausschnitt aus dem IDERepository.kt

```
interface IDERepository{
 fun getVolatileIDEWorkspace(miniWorldProfileModel:
  ↪ MiniWorldProfileModel?, speakResponse: Boolean):
  ↪ Observable<IDEWorkspaceModel?>
 fun getIDEWorkspace(miniWorldProfileModel:
  ↪ MiniWorldProfileModel?, speakResponse: Boolean):
  ↪ Observable<IDEWorkspaceModel?>
 fun addFact(miniWorldProfileModel: MiniWorldProfileModel?,
  ↪ logicItem: LogicItem): Observable<IDEWorkspaceModel?>
 fun addSingleFact(miniWorldProfileModel:
  ↪ MiniWorldProfileModel?, logicItem: LogicItem):
  ↪ Observable<IDEWorkspaceModel?>
 fun addCommonRule(miniWorldProfileModel:
  ↪ MiniWorldProfileModel?, logicItem: LogicItem):
  ↪ Observable<IDEWorkspaceModel?>
 fun askQuery(miniWorldProfileModel: MiniWorldProfileModel?,
  ↪ logicItem: LogicItem): Observable<IDEWorkspaceModel?>
```

Die Methoden des Repositories muss eine entsprechende Klasse im Data-Modul implementieren. Sofern ein Nutzer eine Anfrage an das System stellt, entscheidet die Businesslogik des entsprechenden UseCases, wie mit dieser verfahren werden soll. Gegeben sei bereits eine Anfrage an die Webschnittstelle Dialogflow. Die Antwort muss von den entsprechenden UseCases verarbeitet werden. Quelltext 5.8 zeigt das Auswerten des Dialogflow Intent-Types (einen der relevanten Schritte). Je nach Dialogflow Intent werden die Daten an den entsprechenden Output-Port des Repositories weitergeleitet.

Quelltext 5.8.: Ausschnitt aus buildUseCaseObservable() in AddKnowledgeUseCase.kt

```
override fun buildUseCaseObservable(params: Params):
 ↪ Observable<IDEWorkspaceModel?> {
 return when(params.intentType){
  APIAIIntentType.SIMPLE_FACT ->
          this.ideRepository.addFact(
                  params.miniWorldProfileModel,
                   ↪ params.logicItem)
  APIAIIntentType.COMMON_RULE ->
          this.ideRepository.addCommonRule(
```

```
                params.miniWorldProfileModel,
                 ↪ params.logicItem)
```

5.4.3. Data-Modul

Im Data-Modul werden, wie im App-Modul, Details implementiert. Im Gegensatz zum App-Modul liegt der Fokus eher auf externen Quellen und der Persistierung. Im Rahmen der Arbeit sind hauptsächlich drei Schnittstellen wichtig.

1. Die Kommunikation mit der Dialogflow-API (siehe Abschnitt 5.5). Diese ist in einem ServiceHandler implementiert (siehe Quellcode 5.9)

Quelltext 5.9.: getApiAiResponse() in APIAiServiceHandler.kt

```
override fun getApiAiResponse(result: AIResponse?):
 ↪ Observable<AIResponse?> {
 response = result
 if (response == null) {
  aiService.setListener(this)
  aiService.startListening()
 }
 return Observable.defer({ Observable.fromCallable { response
  ↪ } })
  .retry({ _, error -> error is NullPointerException
 })
}
```

2. Das Übersetzen und Ausführen des Prolog-Quellcodes. Der bestehende Workspace einer Mini-Welt wird in eine für die externe Prolog-Engine interpretierbare Form überführt. Anschließend kann auf dieser Wissensbasis eine Frage getestet werden (siehe Quellcode 5.10)

Quelltext 5.10.: Ausschnitt aus executePrologRequest() in PrologService.kt

```
val theory = Theory(prologMapper
 ↪ .toPrologKnowledgeBaseCode(ideWorkspaceModel
 ↪ .knowledgeBase as
 ↪ List<LogicItem>))
engine.theory = theory
val question = prologMapper.toPrologQuery(logicItem)

val listResult = ArrayList<SolveInfo>()
var result = engine.solve(question)
listResult.add(result)

while (result.hasOpenAlternatives()){
        result = engine.solveNext()
        listResult.add(result)
}
```

3. Das Persistieren der Mini-Welten und die dazugehörigen Workspaces. Um dies zu gewährleisten, verwendet das Data-Modul eigene Model-Objekte (Data-Model-Objekte). Die zu persistierenden Objekte sollen in einer passenden Form vorliegen. Das Transformieren übernimmt wie auch im App-Modul ein Mapper, der die Objekte in beide Richtungen transformiert. Um nicht direkt auf SQLite-Ebene arbeiten zu müssen (Android integrierter Standard), verwendet das Modul ein ORM-Framework, das die Abstraktion von relationalen Tabellen zu Objekten übernimmt. Die DataStore-Klassen implementieren das entsprechende Repository-Interface aus dem Domain-Modul, um dessen Details auf dieser Ebene zu integrieren. Im Quellcode 5.11 ist eine Methode aus dem *MiniWorldsDataStore* zu sehen.

Quelltext 5.11.: addMiniWorldProfile() in MiniWorldsDataStore.kt

```
override fun addMiniWorldProfile(miniWorldProfileModel:
 ↪ MiniWorldProfileModel): Completable {
 return Completable.fromCallable({
  val profile = MiniWorldProfileDataModel()
  profile.description = miniWorldProfileModel.description
  profile.image = 100
```

```
  database.save(profile)
 })
}
```

5.4.4. Reactive Programming zur Überwindung der Modulgrenzen

In den bisher beschriebenen Quellcodebeispielen sind Klassen wie *Single<>*, *Observable<>* oder *Completable* abgebildet. Sie gehören zu dem Reactive Programming Framework *ReactiveX* beziehungsweise dessen Java/Android Ausprägung *RXJava*. Dieses stellt eine API für asynchrones Programmieren mit *observable streams* (siehe Abschnitt 2.2.3) bereit. Den Ablauf möchte ich an einem Anwendungsfall deutlich machen, der mit der User Story 8 (*Als Lernender möchte ich eine Übersicht über alle Mini-Welten haben, um eine gespeicherte auswählen zu können*) zusammenhängt. Das Sequenzdiagramm im Anhang A.18 zeigt den Kontrollfluss schematisch für den Anwendungsfall. Im Diagramm sind lediglich die wichtigsten Aufrufe, die für das Verständnis wichtig sind, aufgeführt.

Die View (Fragment) wird über das Android System aufgerufen und führt *onResume()* aus. Diese wiederum ruft die *resume()*-Methode des Presenters auf. Der Presenter delegiert die nötigen Aufrufe, um der View das View-Model zur Verfügung zu stellen. Hierzu nutzt die Anwendung *RXJava*. Der Presenter wird zum *Observer*, indem er ein Observable-Datenobjekt erwartet und *Consumer*-Funktionen (funktionale Callback-Interfaces von *ReactiveX*) registriert, die bei Änderung des Datenstroms die spezifischen Objekte erhalten (siehe Anhang B.7). Damit nutzt der Presenter den Output-Port des Domain-Moduls. Alle UseCases erben von einer abstrakten Klasse, die die execute-Methode implementiert. In dieser ist das Erstellen des *Observables* zu erkennen (siehe Anhang B.8).
Der konkrete UseCase (in dem Fall *GetMiniWorldsUseCase*) ruft die entsprechende Repository-Methode auf (siehe Anhang B.9), die durch eine Datenbankklasse (*MiniWorldsDataStore*) implementiert wird. Das angefragte Objekt aus der Datenbank wird als Observable-Objekt angeboten. Anschließend gibt die Methode das Objekt an den UseCase zurück, ohne die Closure (Codeblock innerhalb) auszuführen (siehe Anhang B.10). Bisher sind noch keine Datenbanktransaktionen erfolgt. Nachfolgend registriert sich der UseCase auf das Observable-Objekt. Ab hier laufen zwei Stränge parallel. Nach

dem subscribe()-Aufruf geht die Aufruf-Hierarchie der Methoden bis zum Fragment zurück. Der Presenter oder die View könnten weitere Aktionen ausführen (was in dem Beispiel nicht nötig ist). Solange keine Daten vorhanden sind, wird ein Lade-Spinner auf dem Bildschirm angezeigt (siehe Abbildung 5.7). Durch den subscribe()-Aufruf wird

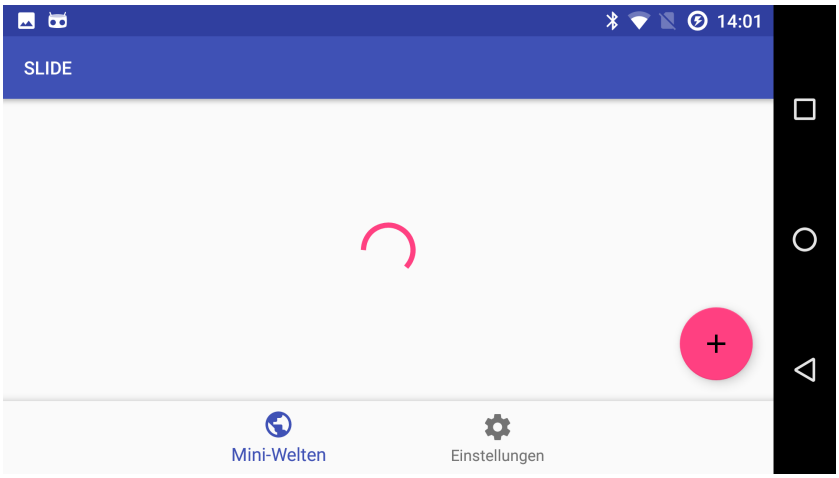


Abbildung 5.7.: App Hauptseite während der Datenbankaufruf stattfindet

das Observable-Objekt aktiviert und führt die Datenbankoperationen aus (siehe Anhang B.10). Sobald das Objekt aus der Datenbank vorliegt, wird es in den Stream gegeben. Der Presenter bzw. die *Consumer*-Callbacks, die auf Änderungen im Datenstream lauschen, werden über das neue Observable-Objekt informiert (siehe Anhang B.7 Zeile 3-5). Der *Consumer* fängt das Objekt aus dem Stream ab. Dieses Model-Objekt wird in ein ViewModel-Objekt überführt und der View übergeben (Zeile 4). Der Spinner verschwindet und die ViewModel-Objekte werden eingefügt (siehe Abbildung 5.8).

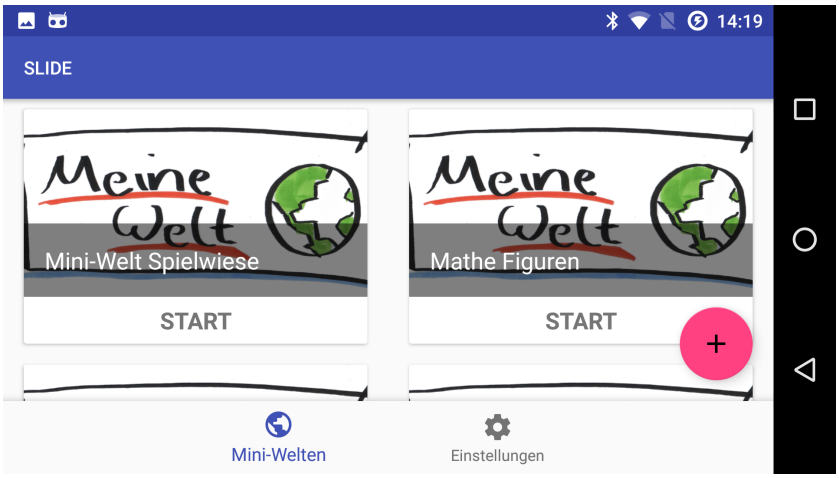


Abbildung 5.8.: Daten aus der Datenbank werden angezeigt

Dieses Beispiel der Kommunikation zeigt noch einmal komprimiert die Stärken von Reac-

tive Programming und des Repository-Patterns innerhalb der Clean Architecture. Durch das Registrieren (subscribe) auf das Observable des entsprechenden UseCases öffnet das Domain-Modul den Output-Port für die *Consumer*. Daten, die durch den UseCase dort verändert werden, gelangen durch eine automatische Benachrichtigung bei Änderungen auf dem Datenfluss wieder zum Presenter, ohne dass die Abhängigkeitsregeln verletzt werden. Die Aktualisierung des Datenflusses geschieht meist durch das Data-Modul (Datenbanktransaktion oder Webschnittstellenaufruf). Dessen Steuerung erfolgt erneut implizit über die Output-Ports *Repository* oder *Handler*. Somit können die Daten innerhalb der Anwendung vom App-Modul über das Domain-Modul in das Data-Modul und wieder zurück zur Oberfläche fließen, ohne dass das Domain-Modul eine *äußere*-Schicht direkt aufruft.

5.4.5. Testen in der Architektur

Das Domain-Modul wurde als reines Java/Kotlin-Modul ohne direkte Abhängigkeiten zu *app* und *data* integriert. Dies hat für das Testen einen immensen Vorteil. Die Businesslogik kann ohne ein Android-Device in reinen Unit-Tests getestet werden. Dies werde ich am einfachen Beispiel der User Story 22 (*Als Lernender möchte ich immer eine Standard-Mini-Welt sehen, um diese als Spielwiese nutzen zu können*) veranschaulichen. Die Spielwiesen-Mini-Welt soll so umgesetzt werden, dass die Inhalte nicht persistent sind. Daher muss eine Unterscheidung für die Datenschicht existieren. Den korrekten Aufruf steuert die Businesslogik in dem entsprechenden UseCase (Interactor). Die JUnit-Testklasse ist in Anhang B.1 einzusehen. Durch das *gemockte* Repository-Interface (Zeile 11), kann sichergestellt werden, dass bei Eingabe des nicht-persistenten Test-Models (Zeile 18) die korrekte Methode aufgerufen wird (*getVolatileIDEWorkspace()*) und keine andere (Zeile 21). Die aus dem Test und der Anforderung resultierende Interactor-Klasse (*GetIDEWorkspaceUseCase.kt*) und deren Methoden sind im Anhang B.2 abgebildet.

5.5. Dialogflow Integration

Die natürlichsprachlichen Eingaben der Nutzer müssen verarbeitet und interpretiert werden. Hierzu bietet Dialogflow die Schnittstelle.

5.5.1. Intents

Das Vorgehen ist ähnlich wie in Abschnitt 2.5.1 beschrieben. Beginnend mit den Intents sind drei mögliche, unterschiedliche Absichten für die Anwendungsfälle in dieser Arbeit identifizierbar.

1. *simple_fact* - Einen Fakt hinzufügen.
2. *common_rule* - Eine Regel hinzufügen.
3. *ask_query* - Eine Frage an die Wissensbasis stellen, die positiv oder negativ beantwortet werden kann.

Das *simple* in *simple_fact* bedeutet, dass es ein Fakt ohne Variable ist. Beispiel: *Max mag Lisa* (in Prolog: *mag(max,lisa).*) statt *Max mag jemanden* (in Prolog: *mag(max, X).*). Das *common* in *common_rule* bedeutet ebenfalls, dass alle Bestandteile der Regel ohne Variablen auskommen. Eine Erweiterung oder Integration der Fakten und Regeln mit Variablen in die bestehenden oder in separate Intents wäre möglich, ist aber nicht Fokus dieser Arbeit.

5.5.2. Entitäten

Entitäten sind im Falle der Entwicklungsumgebung *SLIDE* alle Wörter, die als Konzepte zum Fakt, zur Regel oder zur Frage dazugehören. Da dies potentiell jedes Wort ist, gibt es keine endliche Liste, die gefüllt werden kann. Um Dialogflow das Zuordnen zu erleichtern, sind die Entitäten für den ausgewählten, speziellen Anwendungsfall im Mathematikunterricht (siehe Kapitel 6) eingetragen (siehe Abbildung 5.9). Zu beachten ist vor allem die Checkbox *Allow automated expansions*, die bei Aktivierung eine automatische Erweiterung der Liste ermöglicht. Somit sollte die Schnittstelle auch in der Lage sein, während der produktiven Verwendung weiter zu lernen und sich zu verbessern.

5.5.3. Intents erkennen

Um in den Anfragen die relevanten Entitäten zu erkennen, sind sie mit dem jeweiligen Intent verknüpft. Am Beispiel *simple_fact* (siehe Abbildung 5.10) ist dies zu erkennen. Fakten bestehen immer aus mindestens einem Prädikatsnamen (Prolog: *ecken.*). Daher ist der Parameter *e_out1* als erforderliches Feld markiert. Alternativ bestehen Fakten

Abbildung 5.9.: Dialogflow - Ausschnitt aus der Liste Entitäten

aus dem Prädikatsnamen und Argumenten (in Prolog: *ecken(5).*). Da die Anzahl der Argumente 0 bis n sein kann, wird der Parameter *e_in1* als Liste modelliert.
Die Intents sind sowohl im Example- als auch im Template-Modus mit entsprechenden Sätzen zu füllen (siehe Abbildung 5.11) Die Intents für *common_rule* und *ask_query* sind genauso wie *simple_fact* aufgebaut. Nach dem Modellieren kann Dialogflow trainiert werden, um zuverlässig die gewollten Intents und Entitäten zu erkennen. Durch Nutzung der Schnittstelle und Anpassen der Intents und Entitäten im Trainingsbereich der Weboberfläche müssen Korrekturen oder Bestätigungen per Hand zugeordnet werden. So ist sichergestellt, dass das System kontinuierlich lernen und verbessert werden kann.

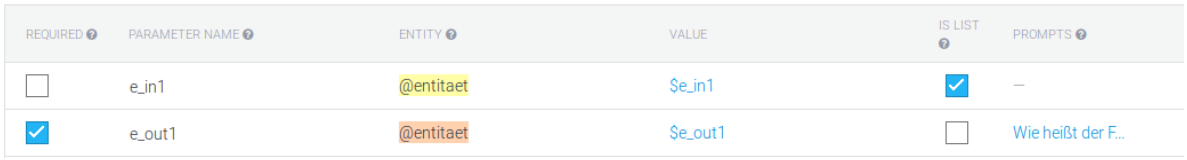


Abbildung 5.10.: Entitäten zu *simple_fact*

Abbildung 5.11.: Nutzer Inputs im Example- und Template-Modus

5.5.4. Kommunikation SLIDE mit Dialogflow

Dialogflow ist ein Onlinedienst, der durch eine Programmierschnittstelle angesprochen werden kann. Somit können verschiedene Clients auf das modellierte NLP-Backend zugreifen. Die App SLIDE nutzt das offizielle SDK (Software Development Kit) für eine Kommunikation. Die Kommunikation zwischen der App und dem Backend kann vereinfacht wie in Abbildung 5.12 dargestellt werden. Es ist ein HTTP-Request auf eine Webschnittstelle. Um die Kommunikation der App SLIDE mit Backend Dialogflow und den groben Ablauf einer Anfrage zu erläutern, ist das Beispiel *Ich sehe fünf Ecken* geeignet. Im Anhang A.17 ist der Verlauf abgebildet. Dabei ist zu beachten, dass die interne App-Kommunikation und die Rückgabewerte vereinfacht dargestellt sind. Zum detaillierten Verständnis des Kontrollflusses über die einzelnen, internen Modulgrenzen der App ist der Abschnitt 5.4.4 heranzuziehen. Der gesprochene Satz (*Ich sehe fünf Ecken*) ist durch eine systemeigene SpeechToText-Engine in eine Zeichenkette konvertiert worden. Diese Zeichenkette wird über den entsprechenden UseCase im Domain-Modul an ein ServiceHandler-Objekt im Data-Modul übertragen (vgl. auch Abschnitt 5.4.3 Beispiel 1). Der durch das SDK gekapselte HTTP-Aufruf an den entsprechenden Dialogflow-Service liefert die Ergebnisse der Anfrage. In diesem Fall wurde erkannt, dass der Nutzer die Ab-

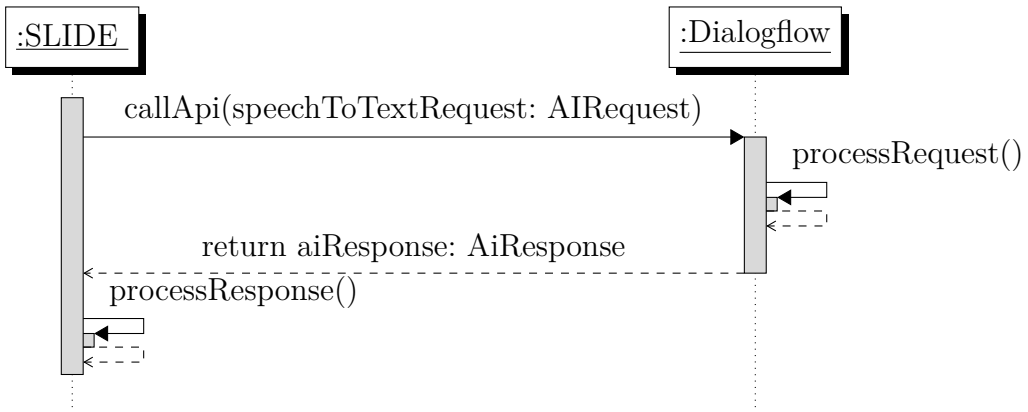


Abbildung 5.12.: Abstrakte Kommunikation zwischen der App und Dialogflow

sicht hat, einen Fakt hinzuzufügen. Dieser Fakt besteht aus dem Prädikatsnamen *Ecken* und dem Prädikatsparameter *5* (in Prolog: *ecken(5)*). Das Domain-Modul verarbeitet das Antwort-Objekt und lässt es durch das Data-Modul persistieren. Im Anschluss holt sich das Domain-Modul einen aktualisierten Workspace (alle Fakten und Regeln). Der Workspace wird durch das App-Modul entgegen genommen und die View aktualisiert sich (siehe Abbildung 5.13).

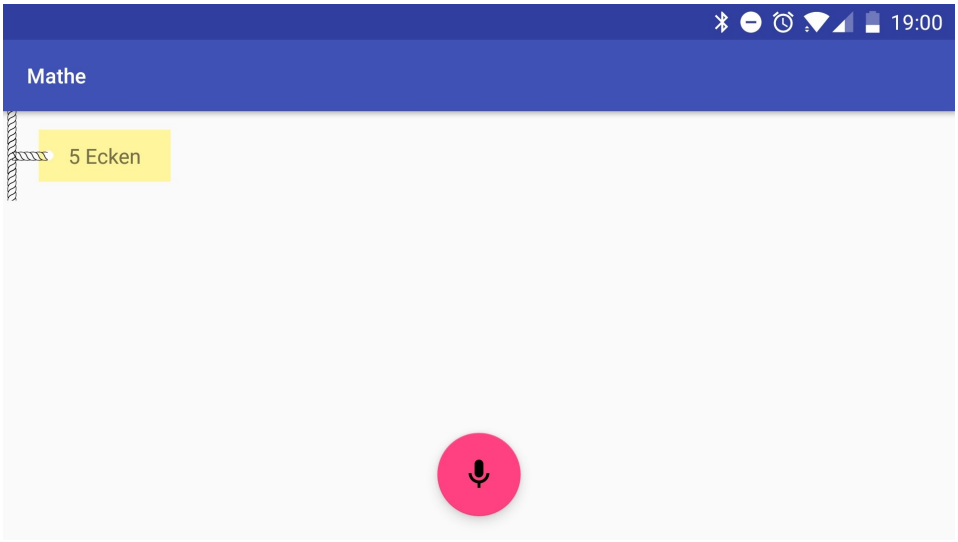


Abbildung 5.13.: Der Fakt *5 Ecken* (in Prolog: *ecken(5)*)

6. Planung der Unterrichtseinheit SLIDE

Dieses Kapitel beschäftigt sich mit dem Einsatz des Werkzeugs *SLIDE* im Schulunterricht. Es wird aufgezeigt, welche Planung, welches Material und Vorgehen zur Beantwortung der Forschungsfragen als Basis ausgearbeitet wurden. Die Schülerinnen und Schüler erlernen neben dem Umgang mit der als Werkzeug fungierenden App auch das Informatikwissen über Expertensysteme mit Fakten und Regeln. Somit stehen neben dem gering gehaltenen Produktwissen über die App besonders das Konzeptwissen über logische Programmierung sowie die fachlichen, mathematischen Kernkompetenzen durch den fachimmanenten Ansatz im Mittelpunkt. Die Unterrichtsstunden sind stark handlungsorientiert geplant.

Es sind zwei Unterrichtsstunden vorgesehen, die sich zu einer Unterrichtseinheit zusammensetzen. Die Lerngruppe besteht aus 12 Schülerinnen und Schülern der vierten Klassenstufe in einer Programmier-AG. Die Planung der Einheit wird in den folgenden beiden Kapiteln vorgestellt und näher erläutert.
Das gewählte fachliche Thema, in dessen Kontext die App SLIDE integriert wird, ist aus dem Niedersächsischen Kerncurriculum Mathematik für die Grundschule entnommen. Das Thema heißt Körper. Unter *Inhaltsbezogene Kompetenzen* aus dem Bereich *Raum und Form* ist nachzulesen, dass die Schülerinnen und Schüler bis zum Ende des Schuljahrgangs 4 unter anderem folgende Kompetenzen erworben haben sollen. „Die Schülerinnen und Schüler erkennen, benennen die geometrischen Körper (Quader [Würfel als besondere Quader], Kugel, Zylinder und Pyramide) und beschreiben ihre Eigenschaften mit Fachbegriffen (Ecke, Seite, Kante, Fläche, senkrecht zueinander, parallel zueinander, rechter Winkel)“ (Niedersächsisches Kultusministerium, 2017, S. 33).
Hinsichtlich des Einsatzes von Medien gibt das Kerncurriculum eine klare positive Empfehlung (siehe Abschnitt 3.4).

6.1. Unterrichtsstunde 1: Fakten, Regeln und die Wissensbasis

Bevor eine sinnvolle Prüfung der Ziele in dieser Arbeit möglich ist, soll die erste Unterrichtsstunde als Vorbereitung dienen (siehe Unterrichtsskizze im Anhang A.6).

6.1.1. Schwerpunktziel der Stunde

Die Schülerinnen und Schüler lernen die Konzepte *Fakt* und *Regel* kennen, indem sie durch Zuordnen zu Beispielen deren Unterschiede identifizieren. Anschließend erlernen sie durch Diskussion untereinander und mit der Lehrperson, dass eine Menge von Fakten und Regeln eine Wissensbasis bilden und wie diese durch Fragen angesprochen werden kann.

6.1.2. Der fachliche Lernstand

Die Konzepte der *Fakten*, *Regeln* und einer Wissensbasis sind neu für die Schülerinnen und Schüler. Eine große Herausforderung beim Verstehen dieser Konzepte wird der fehlende mathematische und informatische Hintergrund sein. Basierend auf der Prädikatenlogik (vgl. Abschnitt 2.2.4) könnte eine Einführung in Fakten und Regeln systematisch einfach gelingen. Da dieses Vorwissen aber nicht vorhanden ist und eine weitreichende Behandlung des Themas aus Zeitgründen nicht möglich ist, sollen die zu vermittelnden Konzepte soweit verdichtet werden, dass im Anschluss eine Grundkenntnis vorhanden sein wird.

6.1.3. Didaktische Überlegungen

Ein Aspekt, den es bei Ziel- und Inhaltsentscheidungen zu beachten gilt, kann durch die Gegenwarts- und Zukunftsbedeutung sowie die Exemplarität und Zugänglichkeit beschrieben werden. Hier bilden Fakten und Regeln einen einfachen Zugang. Neben fachlichen Beispielen aus der Mathematik (Dreieck oder Quadrat) können Beispiele aus dem Elternhaus oder der Schule helfen, da Regeln dort zum Alltag gehören. Daher sind Fakten wie *Alle Seiten sind gleich lang.* oder *Mama erlaubt ein Eis.* genauso in ein zu spielendes Memory (siehe Anhang A.8) eingeflossen, wie auch folgende Regeln. *Es ist ein*

Quadrat, wenn es 4 Ecken hat und alle Seiten gleich lang sind. Max ist glücklich, wenn er ein Eis bekommt. Die Darstellung an einem Seil wurde gewählt, um die Grenzen der Mini-Welt zu repräsentieren. Nur Fakten und Regeln an diesem Seil gehören zur Wissensbasis - alles andere nicht.

Begründung - Repräsentationsebenen

Durch Wechsel der Repräsentationsebenen sollen die Schülerinnen und Schüler einen umfassenden Zugang zum Thema erhalten (vgl. Abschnitt 3.3). Das eingesetzte Memory (siehe Anhang A.9) mit anschließender Aufteilung in eine Wissensbasis an einem Seil spricht durch die handelnde und bildliche Darstellung die enaktive und ikonische Ebene an. Da diese Abbildung einer Wissensbasis mit eigener Syntaktik ebenfalls eine formale Darstellungsform repräsentiert, ist die symbolische Ebene zum Teil bedient. Diese wird aber erst vollkommen mit einbezogen, sobald die Schülerinnen und Schüler Zusammenhänge verbalisieren und Zusammensetzungen begründen.

Begründung - Lehr- und Lerntheorien

Diese Unterrichtsstunde zeigt deutlich das Verbinden der Lehr- und Lerntheorien (vgl. Abschnitt 3.1.1). Durch einen stark handlungsorientierten Ansatz mit dem Memory und dem Legen einer Wissensbasis steht der konstruktivistische Aspekt im Mittelpunkt. Jedoch sind die sozialkonstruktivistischen Überlegungen, vor allem in Bezug auf die hohe Heterogenität der Leistungsmöglichkeiten der Lerngruppe zu beachten. Auch durch das kontinuierliche Wiederholen der Fakten und Regeln während des Memoryspiels (erneutes Nachdenken: Was war eine Regel? Was war ein Fakt?) können Parallelen zur behavioristischen Lehr- und Lerntheorie gezogen werden, vor allem durch die Belohnung in Form der Aufnahme der Memory-Pärchen für das korrekte Zuordnen.

Begründung - Kompetenzen

In der ersten Unterrichtseinheit sollen verstärkt informatische Kompetenzen angesprochen werden, da diese die Grundlage für das Arbeiten mit der App darstellen. In den Bildungsstandards Informatik für die Primarstufe (Gesellschaft für Informatik (GI) e. V., 2017) ist konkret weder logische noch deklarative Programmierung zu finden. Das Aufbereiten von Daten bzw. Informationen und deren Weiterverarbeitung für eine Wissensbasis wird zu den inhaltsbezogene Kompetenzen gezählt. Regeln und Fakten müssen

so angeordnet und kombiniert werden, dass daraus ein bestimmtes Ergebnis resultiert. Die zu verknüpfenden prozessbezogenen Kompetenzen sind gut zu begründen. Während des Memory-Spiels und des Zusammensetzens der Wissensbasis müssen die SuS in Paarungen kommunizieren und kooperieren. Des Weiteren soll die Wissensbasis mit den Kärtchen am Seil dargestellt und begründet werden. Durch Impulse der Lehrperson (Verändern der Wissensbasis und Nachfragen was diese Änderung für Auswirkungen hat) müssen die SuS die gegebenen Fakten und Regeln interpretieren.

6.2. Unterrichtsstunde 2: Körper und Expertensysteme

Fächer in allgemeinbildenden Schulen und auch die Informatik als technisches Fach sollten nicht isoliert gesehen werden. „Sie entfalten ihre Wirkung erst im Zusammenspiel mit anderen Fächern [...]“ (Modrow und Strecker, 2016, S. 44). Somit ist es eine bewusste Entscheidung, fachliche Aspekte aus dem Kerncurriculum eines Unterrichtsfaches mit der Vermittlung von informatischem Kompetenzwissen zu verbinden. „Informatische Probleme sind meist nicht fachimmanent, sondern Informatikmethoden werden eingesetzt, um Probleme aus anderen Gebieten zu lösen“ (ebd., S. 25). Das von Modrow und Strecker (ebd.) angesprochene *Gebiet* ist im Fall dieser Forschung die Mathematik.
Die zweite Unterrichtsstunde beinhaltet das Arbeiten mit der App SLIDE zu einem fachlichen Thema der Mathematik. Die Schülerinnen und Schüler setzen eine Wissensbasis um, die Regeln zu Eigenschaften von Körpern (Pyramide, Kugel, Kegel, Würfel, Quader, Zylinder) enthält. Die Unterrichtsskizze ist im Anhang A.7 einzusehen. Abweichend von Modrow und Strecker (ebd.) sollen die Informatikaspekte in dieser Arbeit als Methode und Werkzeug im Mathematikunterricht fungieren (siehe in Abschnitt 6.2.3 Digitales Werkzeug).

6.2.1. Schwerpunktziel der Stunde

Die Schülerinnen und Schüler lernen die App *SLIDE* kennen. Durch die Nutzung der App sollen die Kinder fachliche Themen zu den Eigenschaften und Unterschieden von geometrischen Körpern verstehen sowie verinnerlichen.
Besonderes Augenmerk legt die Einheit auf die Anforderungsbereiche II (Zusammenhänge herstellen - Das Lösen der Aufgabe erfordert das Erkennen und Nutzen von Zusammenhängen.) und III (Verallgemeinern und Reflektieren - Das Lösen der Aufgabe

erfordert komplexe Tätigkeiten wie Strukturieren, Entwickeln von Strategien, Beurteilen und Verallgemeinern) (vgl. Niedersächsisches Kultusministerium, 2017, S. 15). Zur Prüfung und Reflexion der Inhalte der Stunde bearbeiten die SuS zum Abschluss der Stunde einen Fragebogen (siehe Anhang A.2).

6.2.2. Der fachliche Lernstand

Durch die schulische Arbeitsgemeinschaft Programmieren und durch alltäglichen Kontakt mit mobilen Endgeräten besitzen die Kinder Grundkenntnisse zu allgemeingültigen Bedienkonzepten in Apps. Es wird daher angenommen, dass sie keine größeren Probleme mit der Navigation und der Interaktion auf dem User Interface haben. Die sprachliche Kommunikation mit einem Gerät ist jedoch meist neu oder mindestens ungewohnt. Digitale Assistenten sind noch nicht flächendeckend in den Haushalten vorhanden.
Die Eigenschaften (Ecke, Kante, Fläche) zu den Körpern (Pyramide, Kugel, Kegel, Würfel, Quader, Zylinder) wurden bereits im Mathematikunterricht behandelt. Hier sollte Vorwissen vorhanden sein, das zu Beginn der Unterrichtsstunde am Beispiel der Pyramide (siehe Anhang A.13 und A.14)) reaktiviert wird. Die Eigenschaften sind dem Kerncurriculum (vgl. ebd., S. 33) entnommen. Für eine didaktisch geometrische Begründung verweise ich auf Hattermann et al. (2015) “Geometrie: Leitidee Raum und Form”.
Durch eine Unterrichtsstunde zu den Konzepten *Fakten*, *Regeln* und *Wissensbasis* (siehe Abschnitt 6.1) sind die SuS rudimentär mit den Eigenschaften dieser Konzepte in Berührung gekommen.

6.2.3. Didaktische Überlegungen

Hinsichtlich Gegenwarts- und Zukunftsbedeutung sowie der Exemplarität und Zugänglichkeit ist folgendes festzuhalten. Auf Grund der digitalen Assistenten (Alexa, Google Home, etc.) wird das Sprechen mit Computern in natürlicher Sprache alltäglicher. Durch den Aufbau einer Wissensbasis programmieren die SuS einen Assistenten als Expertensystem, für den sie Wissensträger und Gestalter sind. Dieser zukunftsorientierte Ansatz könnte zunehmen, wenn Nutzer von Alexa, Google Home etc. Regeln natürlichsprachlich für beispielsweise Smarthome-Anwendungen umsetzen. *Alexa öffne Heimkontrolle und füge die folgende Regel hinzu. Wenn jemand in das Wohnzimmer geht, dann schalte das Licht ein.*

Digitales Werkzeug

Die Mathematikdidaktik ordnet die App als mediendidaktisches, digitales Werkzeug (vgl. Schmidt-Thieme und Weigand, 2015, S. 462) ein (siehe auch Abschnitt 3.3). Interessant ist hier der unterschiedliche Ansatz der Mathematikdidaktik im Vergleich zur Informatikdidaktik (siehe Abschnitt 3.2). Letztere benötigt fachliche Aspekte, beispielsweise aus der Mathematik, um so Lösungen hinsichtlich eines realen Problems zu ermitteln. Mathematikdidaktik nutzt solche Medien hauptsächlich, um das Vermitteln mathematischer (und überfachlicher) Aspekte methodisch zu erweitern. Im Rahmen dieser Arbeit wird die App als Werkzeug im Fach Mathematik eingesetzt. Hierbei sollen die SuS, gemäß des didaktischen Dreiecks und des Einsatzes digitaler Technologien (siehe Abschnitt 3.3.1), die Sicht auf Eigenschaften zu Unterschieden und Gemeinsamkeiten bei geometrischen Körpern erweitern. Des Weiteren wird der verstärkte Fokus auf die Versprachlichung und das Kommunizieren gelegt. Die SuS tauschen sich mit dem Partner aus und versprachlichen die Fachbegriffe und Zusammenhänge wiederholend und kontinuierlich, indem sie in die App sprechen.

Methodik - Ablauf

Die Unterrichtstunde startet mit der Motivation und Zielformulierung in der Arbeitsform *Kinositz* (Stuhlhalbkreis vor der Tafel). Den SuS wird nochmals in Erinnerung gerufen, was Programmieren (für ihren Kontext) bedeutet, nämlich dass Computern etwas beigebracht wird. Ziel ist es, einer Wissensbasis (Mini-Welt) die Eigenschaften der Körper beizubringen. Um zu Beginn eine direkte Verknüpfung zur vorangegangenen Stunde zu schaffen, wird die Eingangsmotivation über ein Beispiel aus dem Memory (siehe Anhang A.9) hervorgerufen. Die Lehrperson zeigt die App *SLIDE* mit deren Wissensbasis (siehe Abbildung 6.1) und demonstriert die Auswirkungen des Faktes *Sonne scheint*. Die SuS sehen die ersten sprachlichen Interaktionen mit der App (Fakten hinzufügen, Fakt löschen, Wissensbasis anfragen).

Der fachlich mathematische Einstieg erfolgt im Anschluss. Nach der Aktivierung des Vorwissens mit Hilfe des Pyramidenplakats (siehe Anhang A.13 und A.14) sollen die SuS die Eigenschaften des Körpers als Fakten nennen. Nachdem die Lehrperson den ersten Fakt der Wissensbasis hinzugefügt hat, dürfen einzelne SuS die verbleibenden Fakten hineinsprechen. Die Regel wird durch die Lehrperson exemplarisch beschrieben. Abschließend stellen die SuS Fragen an die Wissensbasis. *Ist es eine Pyramide? Ist es eine Kugel?*

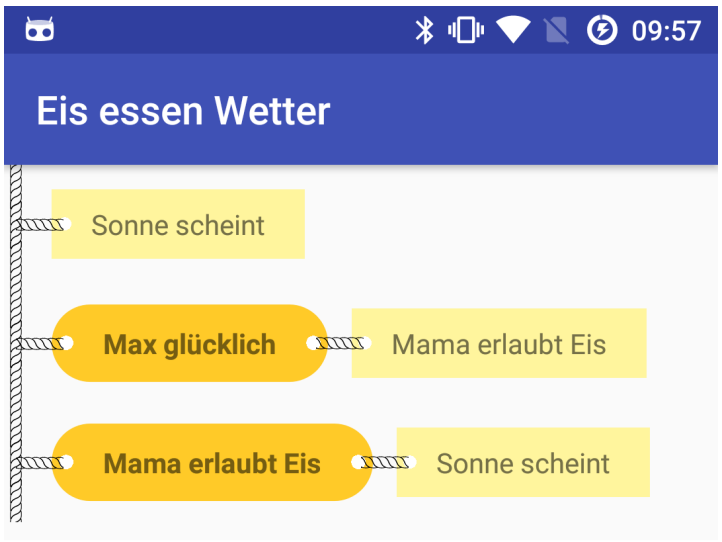


Abbildung 6.1.: Screenshot der Mini-Welt, ob Max glücklich ist.

In der nachfolgenden Phase wird die Arbeitsform *Lerntheke* genutzt. In Gruppen arbeiten die SuS an jeweils einem Körper. Die Materialien (Aufgabenbeschreibung des jeweiligen Körpers (siehe Anhang A.12), Hilfezettel zur Bedienung der App (siehe Anhang A.10), der Laufzettel (siehe Anhang A.11) sowie der jeweilige Körper (siehe Anhang A.15)) müssen von einer *Theke* abgeholt werden. Sobald ein Körper fertig bearbeitet wurde, sind die Materialien zu dem Körper auf die *Theke* zurückzulegen. Der Laufzettel wird abgehakt und die Materialien für einen neuen Körper werden mitgenommen.

Syntaktische Hürden

> *„Kommunikation mit Maschinen ist im Wesentlichen ein Syntaxproblem, sie erfordert sprachliche Präzision.“* ((Modrow und Strecker, 2016, S. 105))

Die App *SLIDE* soll den Lernenden als *mächtiges Werkzeug* (vgl. ebd., S. 25) unterstützen. Sowohl die Informatik-Konzepte einer logischen Programmiersprache (siehe Abschnitt 2.2.4) als auch die fachlichen Aspekte im Schulunterricht sollen gefördert und erlernt werden. Durch das Herabsetzen der syntaktischen Hürden mit Hilfe der natürlichen Sprache und deren Interpretation durch NLP (siehe Abschnitt 2.5), soll es den Schülerinnen und Schülern möglich sein, diese Anwendung zielgerichtet einzusetzen und daran zu lernen.

Begründung - Repräsentationsebenen

Die Repräsentationsebenen und der methodische Wechsel zwischen ihnen ist in dieser Unterrichtsstunde gut abzugrenzen. Die eingesetzten haptischen Körper und das Bedienen der App sprechen die enaktive Repräsentationsebene an. Durch eine syntaktische Darstellung (siehe Abschnitt 5.3.2) der Eigenschaften von Körpern durch die Fakten und Regeln in der Mini-Welt der App ist ein Wechsel auf die ikonische Ebene möglich. Die gewählte Darstellung an einem Seil und dessen Bedeutung kennen die SuS bereits aus der vorangegangen Unterrichtsstunde (siehe Abschnitt 6.1). Die symbolische Ebene erhält durch den Einsatz dieser App als Werkzeug im Kontext digitaler Technologien und Medien eine besondere Rolle. Zum einen müssen die Gruppenmitglieder vorher diskutieren und sich absprechen, welcher Fakt oder welche Regel als nächstes der Wissensbasis hinzugefügt werden soll. Zum anderen versprachlichen sie das Ergebnis der Diskussion beim Hinzufügen. Hier sehe ich einen, wie von Schmidt-Thieme und Weigand (2015) (vgl. Abschnitt 3.3.1) angesprochen, besonderen Mehrwert durch den Einsatz der App.

Begründung - Lehr- und Lerntheorien

Diese Unterrichtsstunde zeichnet sich durch einen stark handlungsorientierten Ansatz aus (vgl. Abschnitt 3.1.1). Die SuS entscheiden selbst, welchen Körper sie als nächstes bearbeiten. Sie entdecken mit diesem Körper dessen Eigenschaften unter Bezugnahme der App mit eigenen natürlichsprachlichen Eingaben. Durch das Arbeiten in Paaren und das Übernehmen des Wissens oder Vorgehens von anderen ist ebenfalls ein sozialkonstruktivistischer Lernansatz erkennbar Die behavioristische Lehr- und Lerntheorie wird durch das wiederholte Formulieren der Fakten und Regeln deutlich, die die Eigenschaften (Wie viele Ecken nochmal?) und Fachbegriffe (Ecke, Kante, Fläche) enthalten.

Begründung - Kompetenzen

Da die App in dieser Unterrichtsstunde als Werkzeug im Kontext der Mathematik eingesetzt wird, müssen die Bildungsstandards dieses Faches zur Begründung der Kompetenzen herangezogen werden. Neben dem beschriebenen Einsatz zu Medien im Kerncurriculum Mathematik (vgl. Niedersächsisches Kultusministerium, 2017, S. 14) und den inhaltsbezogenen Kompetenzen zu Körpern (vgl. ebd., S. 33) sind vor allem die prozessbezogenen Kompetenzen Darstellen und Kommunizieren im Mittelpunkt (siehe Abschnitt 3.4). Das Darstellen ist über das Ordnen der Eigenschaften zu Fakten bzw.

Regeln und dessen Repräsentation in der App geschehen. Hierbei spielt besonders der Einsatz digitaler Technologien eine besondere Rolle (vgl. Abschnitt 3.3.1). Die Kompetenz Kommunizieren wird vor allem durch das Miteinander mit dem Partner forciert, denn es muss besprochen werden, was als nächstes der Wissensbasis hinzuzufügen ist. Dies soll den Austausch über mathematische Sachverhalte und deren Verständnis fördern (vgl. ebd., S. 7). Darüber hinaus kommunizieren die SuS mit der App und überprüfen die Eingaben und Antworten.

6.3. Integration der geplanten Stunden in eine bestehende Unterrichtseinheit

Im Rahmen dieser Arbeit werden die beiden Unterrichtsstunden zu Forschungszwecken isoliert vom Mathematikunterricht durchgeführt. Sie sind allerdings als integrierter Bestandteil der Unterrichtseinheit *Körper* angedacht. Da die App als Entwicklungsumgebung konzipiert ist, soll sich der Anwendungsfall nicht lediglich auf den einen hier vorgestellten Aspekt beschränken. Ein Einsatz in anderen Unterrichtseinheiten und Fächern ist genauso denkbar. Dies müsste allerdings in einem separaten Rahmen in einer eigenen Forschung mit eben diesem Schwerpunkt untersucht werden.
Ein Beispiel für eine klassische Unterrichtseinheit zum Thema Körper ist im Anhang A.5 zu finden. Bei bereits bekanntem Basiswissen zu Fakten, Regeln und den Wissensbasen (siehe Abschnitt 6.1) - dieses Wissen wäre bei Mehrfachnutzung/Integration vorhanden - könnte die Unterrichtsstunde *Körper und Expertensysteme* (siehe Abschnitt 6.2) sinnvoll an zwei Stellen eingebaut werden. So ist eine Möglichkeit die zweite Unterrichtsstunde aus der klassischen Einheit (siehe Entwurf Anhang A.5) mit Hilfe der App durchzuführen, da diese das Kennenlernen der Eigenschaften als inhaltlichen Schwerpunkt setzt. Eine weitere Möglichkeit besteht darin, die App als Wiederholung und Prüfungsvorbereitung am Ende der Unterrichtseinheit einzusetzen.

7. Forschungsmethode

In diesem Kapitel geht es um die Beschreibung der Arbeits- und Forschungsmethode für die Erhebung der Daten, auf deren Grundlage die Überprüfung der in Kapitel 4 aufgestellten Ziele und Fragestellungen stattfindet. Darüber hinaus werden die erarbeiteten und erstellten Hilfsmittel für die Erhebung erläutert und begründet. Aufgestellte Bewertungskriterien bilden den Abschluss dieses Kapitels. Diese werden den Forschungsfragen zugeordnet, um eine spätere Bewertung nachvollziehbar darzustellen.

7.1. Qualitative Sozialforschung

Im Zusammenhang mit den Zielen und Fragestellungen einer Forschungsarbeit ist die Wahl der richtigen Methode zentral. In der Vergangenheit wurde vor allem der quantitativen Sozialforschung besondere Aufmerksamkeit gewidmet. Vorbild sei hier vor allem die Naturwissenschaft im Zusammenhang von Ursache und Wirkung gewesen (vgl. Flick, 2011, S. 23f). Jedoch werde heutzutage die qualitative wie die quantitative Methode als valides, eigenständiges und wichtiges Forschungswerkzeug angesehen. Flick (ebd.) fasst hier Quellen zusammen und stellt heraus, dass nunmehr, trotz ihrer verschiedenen und gleichberechtigten Ansätze, eine Kombination beider Strategien je nach Zielformulierung erstrebenswert und sinnvoll sein kann. Den Hauptunterschied beschreibt Brüsemeister (2008, S. 19) als „*überprüfende* [quantitativ] versus *entdeckende* [qualitative] Forschungslogik“. Das bedeutet, dass quantitative Forschung eher dann Anwendung finde, wenn die theoretische Grundlage bereits vorhanden sei, aber ein messbarer Erweis noch erbracht werden muss. Voraussetzung sei eine hohe und repräsentative Anzahl an Datensätzen. Somit beschreibt sie eher die „Aggregation bestimmter Variablenmerkmale, als statistische[n] Zusammenhang“ (ebd., S. 20). Qualitative Forschung hingegen legt Wert auf unterschiedliche Perspektiven, sowie die Reflexion des Durchführenden und dessen Erkenntnisse. Diese persönlichen Bewertungen werden damit Teil der Forschung (vgl. Flick, 2011, S. 26f). Somit könne nach Flick (ebd.) auch dann empirische Forschung betrieben

werden, wenn die Zusammenhänge eher komplexer Natur seien und/oder sich keine eindeutigen Ursache-Wirkungs-Zusammenhänge an unabhängigen, messbaren und isolierten Variablen festmachen ließen. „Gegenstände werden dabei nicht in einzelne Variable zerlegt, sondern in ihrer Komplexität und Ganzheit in ihrem alltäglichen Kontext untersucht. Deshalb ist ihr Untersuchungsfeld auch nicht die künstliche Situation im Labor, sondern das Handeln und Interagieren der Subjekte im Alltag “ (Flick, 2011, S. 27). Dieser qualitative Ansatz der empirischen Forschung passt am besten auf die aufgestellten Ziele und Fragestellungen dieser Arbeit (vgl. Kapitel 4). Somit sind folgende Methodiken aus diesem Bereich ausgewählt, um die Fragen zu beantworten.

7.1.1. Das Leitfaden-Interview

Das Leitfaden-Interview ist eine Technik aus der qualitativen Sozialforschung, die immer größere Aufmerksamkeit erfahre und vielfach eingesetzt wird. Dies rühre daher, dass in der verhältnismäßig offenen Gestaltung (im Gegensatz zu Fragebögen) Meinungen und Sichtweisen der Person oder Personengruppen eher zur Geltung kommen (vgl. ebd., S. 194). Der Leitfaden diene hierbei als Gestaltungsinstrument für den Verlauf des Gesprächs. Dies habe den Vorteil, das Gespräch bei möglichst großer Offenheit am Thema des Forschungsinteresses zu halten (vgl. Helfferich, 2014, S. 560). Helfferich beschreibt weiter, dass der Leitfaden nach dem Prinzip methodisch konzipiert sein sollte: „So offen wie möglich, so strukturierend wie nötig“ (ebd., S. 560).

Experten-Interview

„Leitfadeninterviews gestalten die Führung im Interview über einen vorbereiteten Leitfaden, Experteninterviews sind definiert über die spezielle Auswahl und den Status der Befragten“ (ebd., S. 559). Diese ausgewählte Form des Leitfaden-Interviews soll helfen, die Forschungsfragen zu beantworten. Es wird eine Person ausgewählt, die Experte für ein bestimmtes Handlungsfeld ist. Somit tritt die Person selbst in den Hintergrund. Vielmehr ist die Fachexpertise entscheidend und kann repräsentativ für eine bestimmte Personengruppe angenommen werden (vgl. Flick, 2011, S. 214). Für die Auswahl eines Experten kann die Definition von Bogner, Littig und Menz (2013) herangezogen werden.

> „Der Experte verfügt über technisches, Prozess- und Deutungswissen, das sich auf sein spezifisches, professionelles oder berufliches Handlungsfeld be-

> zieht. Insofern besteht das Expertenwissen nicht allein aus systematisiertem, reflexiv zugänglichem Fach- oder Sonderwissen, sondern es weist zu großen Teilen den Charakter von Praxis- und Handlungswissen auf, in das verschiedene und durchaus disparate Handlungsmaximen und individuelle Entscheidungsregeln, kollektive Orientierungen und soziale Deutungsmuster einfließen“. (Bogner, Littig und Menz (ebd., S. 46))

Demnach ist im Rahmen dieser Arbeit Experte für den didaktischen Bereich die entsprechende Lehrkraft im Fachgebiet. Das Experten-Interview wird somit mit einer beisitzenden Lehrkraft im Anschluss an die Unterrichtsstunden geführt. Meuser und Nagel (2013) identifizieren Probleme, die es bei einem zu offenen Interview geben könnte und stellen daher die Wichtigkeit eines Interview-Leitfadens heraus.

> „Die in die Entwicklung eines Leitfadens eingehende Arbeit schließt aus, dass sich der Forscher als inkompetenter Gesprächspartner darstellt. [...] Die Orientierung an einem Leitfaden schließt auch aus, dass das Gespräch sich in Themen verliert, die nichts zur Sache tun, und erlaubt zugleich dem Experten, seine Sache und Sicht der Dinge zu extemporieren[1]“.
>
> (Meuser und Nagel (ebd., S. 77))

Der ausgearbeitete Leitfaden für das Interview mit der Lehrkraft zu dieser Arbeit befindet sich im Anhang A.1.

7.1.2. Der Fragebogen - Ein Bogen mit Fragen

Eine weitere Erhebungsform ist der Fragebogen, der zur Befragung der SuS genutzt werden soll. „Fragebogen sind [...] zur Selbst[...]beschreibung oder auch zur Selbst[...]-beurteilung einsetzbar. Fragebogen können sich auf unterschiedliche Merkmalsbereiche beziehen und sind nicht auf die Messung von psychischen Merkmalen wie Emotionen oder Persönlichkeitsmerkmalen beschränkt“ (Kallus, 2016, S. 18). Somit ist es möglich, neben Fragen zu emotionalen Merkmalen, ebenfalls ein Teil des fachlich erworbenen Wissens bei den Schülerinnen und Schülern zu überprüfen. Dies hat zur Folge, dass sich der zu konzipierende Fragebogen in zwei Themenbereiche unterteilt. Im ersten Teil sollen die Schülerinnen und Schüler, ähnlich einer Lernzielkontrolle, fachliche Aufgaben lösen. Im zweiten Teil schließt sich ein klassischer, empirischer Fragebogen an. Dieser enthält Frage/Antwort-Items, die sich auf das Arbeiten mit der Anwendung beziehen.

[1] hier: spontan auszudrücken

Das Befragen der Schülerinnen und Schüler wäre prinzipiell ebenfalls in Form eines Interviews möglich. Der Ansatz einen qualitativen Fragebogen zu entwickeln, ist darin begründet, dass die Probandengruppe mit 12 Personen recht groß ist. Dieser Fragebogen soll somit ein qualitatives Interview mit jedem einzelnen Lernenden ersetzen, denn hinsichtlich der Fragestellungen ist es durchaus relevant, wie die einzelnen Schülerinnen und Schüler das Arbeiten mit der App einschätzen. Der Ansatz, aus dem Fragebogen quantitative Ergebnisse zu ermitteln, kann auf Grund der dafür zu geringen Gruppengröße und keiner repräsentativen Auswahl nicht sinnvoll verfolgt werden. Die Gütekriterien (*Objektivität*, *Zuverlässigkeit* und *Validität* (vgl. Kallus, 2016, S. 15ff)) für die zu erfassenden Merkmale könnten somit nicht erreicht werden. Bei der Konzeption der Items und des Fragebogens als Gesamtwerk sind daher ebenfalls bewusst nicht alle Kriterien und Schritte (vgl. ebd., S. 14f) beachtet worden, die für einen psychometrisch geprüften Bogen nach ISO 10075 oder DIN 33430 relevant wären.
Durch den qualitativen Ansatz wird der *Fragebogen* nach Kallus (ebd., S. 15) zum *Bogen mit Fragen*. Nichtsdestotrotz können die Hinweise zum Erstellen der Fragen-Items für quantitative Fragebogen auch für Bogen mit Fragen sinnvoll sein, denn „sowohl bei der qualitativ orientierten Befragung als auch bei den standardisierten Verfahren sollten Fragen und die zugehörigen Antworten so gestellt sein, dass jede/r Antwortende die zutreffende Antwortkategorie unmittelbar angeben kann“ (ebd., S. 132). Kallus führt Beispiele hierfür an. Zu vermeiden seien unterschiedliche Dimensionen innerhalb einer Frage/Antwort. Dieser Fehler sei sowohl in der quantitativen als auch in der qualitativen Befragung zu beobachten. Fragen müssen darüber hinaus differenziert formuliert sein, sodass der Antwortende nicht von der Menge unnötiger Fragen überfordert und *belästigt* werde. Wichtig sei auch, die Befragten über den Sinn der Befragung zu unterrichten. Items sollten zudem frei von Suggestion sein, um eine Beeinflussung zu vermeiden (vgl. ebd., S. 132f). Der erstellte Bogen mit Fragen ist im Anhang A.2 verfügbar.

Gruppenaufnahmen

Da Interviews mit den Schülerinnen und Schülern aus den genannten Gründen nicht sinnvoll sind, wurde der Fragebogen entwickelt. Für die Beantwortung der Forschungsfragen ist es darüber hinaus von Interesse, wie die Gruppenmitglieder miteinander interagieren, diskutieren und die App bedienen, sodass durch Stichproben in zwei der Gruppen weitere Erkenntnisse dokumentiert werden. Diese Dokumentation in Form von Audioaufnahmen der Gespräche können im Anschluss an die Unterrichtsstunden als weiteres Erhebungsinstrument und Methodik Antworten der Schülerinnen und Schülern aus den

Fragebogen unterstützen.

7.2. Transkription

Mit der Transkription erstellt der Forschende ein Transkript. Forschungsfragen, die durch qualitative Rohdaten wie Tonaufnahmen aus Interviews oder Gruppendiskussionen beantwortet werden sollen, müssen in schriftlicher Form vorliegen. Dieser Vorgang heißt in der qualitativen Sozialforschung *Transkription* (vgl. Fuß und Karbach, 2014, S. 15). Als Produkt ergibt sich das Transkript, das „die zentrale Ausgangsbasis der wissenschaftlichen Analyse [ist]“ (ebd., S. 15).
Es existieren mehrere Systeme wissenschaftlicher Transkripte. Welches das passende ist, lässt sich durch die Frage nach der nötigen Detailtiefe beantworten. „Liegt der Fokus der Auswertung auf der Inhaltsebene, so kann das Regelsystem der Transkription gegebenenfalls reduziert werden, indem beispielsweise auf die Verschriftlichung der interaktiven Phänomene, wie Zuhörersignale oder Partiturschreibweise verzichtet [wird]“ (ebd., S. 57). Durch das vorherige Benennen der verschiedenen Transkriptionsregeln (Nutzen von Modulen) wird klar, in welchem Detailgrad das Transkript verfasst wurde. Ein Modul ist eine Transkriptionsregel, die sich mit einem oder mehreren Aspekten der Verschriftlichung befasst. Eine für diese Arbeit relevante Auswahl der Module nach Fuß und Karbach (vgl. ebd., S. 37ff) sind im Anhang A.19 verfügbar.
Im Experteninterview ist besonders die Inhaltsebene entscheidend, aus diesem Grund wird von einer Verwendung von Modulen wie *Interaktion* abgesehen. Da beim Arbeiten mit der App neben anderen Aspekten auch die Kommunikation im Vordergrund steht, ist dies hierbei sehr wohl von Bedeutung. Das transkribierte Material ist im Anhang verfügbar (siehe Anhang C.1, C.2, C.3).

7.3. Fallstudien

Um die Frage 3 *Wie kann die App intern strukturiert sein, um zukünftige Fortschritte bei Frameworks und Technologien zur einfachen Benutzerinteraktion ohne hohe Änderungsaufwände in der App nutzen zu können?* argumentativ beantworten zu können, werden Fallstudien verwendet. Anhand von drei potentiellen, nachträglichen Anforderungen ist zu untersuchen, welche Änderungen und Einflüsse eine Umsetzung auf die App hätte. Fallstudie 1 bezieht sich auf eine Änderung der Anforderung, die alle Module und Ebenen

betrifft. Fallstudie 2 ist thematisch eher am Backend orientiert. Fallstudie 3 beschäftigt sich mit Änderungen am Frontend und der View. Somit wird eine differenzierte Auswahl getroffen, an der die Fragestellung untersucht wird.

7.3.1. Fallstudie 1

User Story 5 - *Als Lernender möchte ich der Mini-Welt Fakten hinzufügen, um diese zu erweitern.* Ein konkretes Akzeptanzkriterium zum Verhalten des Buttons zur Sprachaufnahme war nicht vorhanden. Die Funktionsweise des Buttons wurde dem Dialogflow-SDK angepasst, das eine eigene Implementierung zum Starten und Stoppen der Aufnahme bereitstellt. Das Aufnehmen der Stimme startet beim Tippen auf den Button und stoppt automatisch, sobald das SDK eine Sprachpause feststellt. Aus Messenger-Apps ist eine andere Funktionsweise bekannt. Hier hält der Nutzer den Aufnahme-Button so lange gedrückt, wie die Stimme aufgenommen werden soll. Eine entsprechende Anpassung der Anforderung wäre denkbar. Folgende Akzeptanzkriterien sind aufzunehmen.

- Beim Drücken auf den Aufnahme-Button startet die Aufnahme
- Beim Loslassen des Aufnahme-Buttons stoppt die Aufnahme
- Das Gesprochene soll in Text umgewandelt werden
- Der Text wird an das NLP-Backend gesendet

Der Schnittstellenaufruf zum Dialogflow geschieht konform zur Architektur im data-Modul (siehe Abschnitt 5.4.3). Der generelle Aufruf aus einer View ist an einem Beispiel bereits im Abschnitt 5.4.4 beschrieben. Der dort gezeigte Anwendungsfall verfolgt die identischen Prinzipien und ist übertragbar. In der View wird eine Presenter-Methode aufgerufen (siehe Quelltext 7.1).

Quelltext 7.1.: setOnClickListener in IDEActivity.kt auf dem Aufnahme-Button (fab)

```
fab.setOnClickListener {
 ...
 idePresenter.performUserInput()
```

Über das domain-Modul (*ExecuteUserRequestUseCase*) und dessen Interface-Aufruf (*getApiAiResponse()* in *APIAIHandler*) gelangt die Aktion in die Interface implementierende Klasse *ApiAiServiceHandler* (siehe Anhang B.11 Zeile 7). Dort ist zu erkennen (Zeile 1), dass die Klasse neben dem eigenen *APIAIHandler* den *AIListener* implementiert. Dies ist ein SDK-Interface, das zusätzlich zur SpeechToText-Funktionalität den

resultierenden Text direkt an das Backend sendet (Zeile 10-11). Die Antwort wird im *onResult()* (Zeile 19-21) zurückgegeben. In Anhang C.9 sind die Schritte beschrieben, die für eine Anpassung an die Akzeptanzkriterien nötig wären.

7.3.2. Fallstudie 2

Austausch der NLP-Schnittstelle. Hinsichtlich der Fragestellung und der sich schnell weiterentwickelnden Techniken und Frameworks im Bereich NLP ist es naheliegend, im Gedankenexperiment dessen derzeitige Implementierung auszutauschen. Gegeben sei ein neuer Dienst, der Intentionen und Entitäten erkennt. Dieser soll die Dialogflow-Schnittstelle ablösen. Hierbei ist es irrelevant für diese Fallstudie, ob dies ein Onlinedienst ist oder eine eigens implementierte Offline-Lösung. Anhang C.11 zeigt die Quellcodestellen, an denen eine solche Anpassung umgesetzt werden müsste.

7.3.3. Fallstudie 3

Änderung der grafischen Benutzerschnittstelle. Hinsichtlich der Fragestellung, die unter anderem die einfache Benutzerinteraktion in den Vordergrund stellt, sollte eine Fallstudie die grafische Benutzerschnittstelle abdecken. Ein grundsätzlich anderes Oberflächen-Framework kann auf Grund der Android-Umgebung nicht gewählt werden. Die Darstellung der Mini-Welt (UserStory 19 *Als Lernender möchte ich Änderungen der Mini-Welt beim erneutem Aufrufen wiedersehen, um nicht jedes Mal die gleichen Eingaben wiederholen zu müssen*) ist zu verändern. In Form von farbigen Puzzleteilen sollen die Fakten und Regeln einzeln ineinandergreifen. Anhang C.12 zeigt die Quellcodestellen, an denen Anpassungen stattfinden müssen. Es wurde davon abgesehen, dies zu implementieren, da es vor allem darum geht, die Auswirkungen von Änderungen einzuschätzen.

7.4. Bewertungskriterien

In diesem Abschnitt werden, aufgeteilt auf die Forschungsfragen und Ziele der Arbeit (siehe Kapitel 4), Bewertungskategorien aufgestellt, die im Rahmen der Auswertung zu prüfen sind. Die erstellten Materialien (siehe Abschnitt 7.1.1 und 7.1.2) orientieren

sich an diesen Kategorien. Nachfolgend werden die Kriterien durch eine Nummer (Forschungsfrage) und einen Buchstaben (Kategorie) eindeutig identifiziert. Beispiel Kriterium (K) aus der Frage 2c : K2c

1. *Ist es möglich in einer App die Intention durch natürlichsprachliche Eingaben von Kindern der 4. Klasse zu einem fachlichen Thema korrekt zu interpretieren?*

(a) Intentionen erkennen

(b) Entitäten erkennen

2. *Kann eine App mit einer HMI ausgestattet werden, die Kindern der 4. Klasse die Interaktion mit einem Prolog-Interpreter ohne syntaktische Hürden ermöglicht?*

(a) Verständnis App Bedienung

(b) Motivation beim Bearbeiten

(c) Anwenden der Fakten in gelernter Form

(d) Anwenden der Regeln in gelernter Form

3. *Wie kann die App intern strukturiert sein, um zukünftige Fortschritte bei Frameworks und Technologien zur einfachen Benutzerinteraktion ohne hohe Änderungsaufwände in der App nutzen zu können?*

(a) Software Produktqualität Wartbarkeit - Modularity

(b) Software Produktqualität Wartbarkeit - Modifiability

(c) Software Produktqualität Wartbarkeit - Testability

4. *Hilft die App mit logischem Programmierparadigma und im erarbeiteten Unterrichtsrahmen Kindern der 4. Klasse, fachliche Zusammenhänge sprachlich zu formulieren? Werden die gewünschten Kompetenzbereiche angesprochen?*

(a) Prozessbezogener Kompetenzbereich Argumentieren

(b) Prozessbezogener Kompetenzbereich Kommunizieren

(c) Inhaltsbezogene Kompetenzbereiche: Mathematisches Verständnis Eigenschaften/ Unterschiede der Körper

8. Auswertung

In diesem Kapitel steht die Auswertung der Forschungsergebnisse im Vordergrund. Es werden zum einen die Aspekte der Sozialforschung untersucht, wie auch die Fallstudien. Aufgeteilt in Abschnitte beschäftigt sich die Beschreibung der Ergebnisse mit der sachlichen Darlegung von Daten aus den Fragebogen, Gesprächen und technischen Artefakten. Nachfolgend werden diese Daten anhand der Bewertungskriterien (siehe Abschnitt 7.4) analysiert und interpretiert. Abschließend fasst eine Reflexion die Ergebnisse zusammen.
Die einzelnen Kriterien der Fragen wurden nach der Datenerhebung jeweils induktiv durch eine Abstufung ihrer Ausprägung erweitert. Um Tendenzen erkennen zu können und die Übersichtlichkeit zu gewährleisten, soll es genügen den Grad der Erfüllung in drei Bereiche zu untergliedern. Beispiel K2c: *Hohe Motivation*, *Mittlere Motivation*, *Niedrige Motivation*. Beispiel K3b *Starke Abhängigkeit*, *teilweise Abhängigkeit*, *keine Abhängigkeit*

8.1. Sozialforschung

Dieser Abschnitt beschäftigt sich mit den Beobachtungen, der Beschreibung und Auswertung der Fragen 1,2 und 4 und deren Artefakten. Diese wurden den Fragestellungen, Kriterien und der jeweiligen Ausprägung zugeordnet (siehe Anhang C.9). Hierbei sind die verwandten Fragen (1, 2, 4) im Kontext der Sozialforschung betrachtet. Frage 3 wird nachfolgend separat behandelt (siehe Abschnitt 8.2).

8.1.1. Beschreibung

An der Erhebung nahmen zehn Schülerinnen und Schüler teil. Die folgende Tabelle 8.1 nimmt Bezug auf jedes Kriterium in der Auswertungstabelle (siehe Anhang C.9) und

fasst die Beschreibungen zusammen. Die von den SuS ausgefüllten Fragebogen sind auf dem Datenträger beigefügt (siehe Anhang D.2).

Id	**Zusammenfassung**
K1a	Viele Beobachtungen während der Einheit (siehe Anhang C.2, C.3) zeigten, dass Intentionen korrekt verstanden wurden. Dies wird durch die Log-Einträge des Dialogflow-Backends unterstützt (siehe Anhang D.3). Einige Aussagen wurden jedoch nicht korrekt verstanden.
K1b	Entitäten wurden größtenteils richtig zugeordnet. Bei null wurde in der einen Gruppe *nur* erkannt. Bei einem Satz wie dem folgenden wurde Zylinder als einzig wichtige Entität erkannt. Ha:::llo::: ist es ein Zylinda:::. - Ha! Ich hab sogar hallo gesagt.
K2a	Bei den meisten SuS gab es keine Probleme beim Bedienen der App. 9 von 10 SuS gaben an, dass sie *eher gut* (3) oder *gut* (6) mit der App zurecht gekommen sind. Ein/e Schüler/in gab *schlecht* an. Die meisten SuS (8 von 10) stimmten der Aussage eher oder ganz zu, dass sie jederzeit wussten, was sie drücken müssen. 5 SuS stimmten dieser Aussage ganz zu (*ja*), 3 *eher ja*, ein/e *eher nein* und ein/e *nein*. Den Umstand, dass bei der Spracheingabe in die App der Aufnahme-Knopf nicht gedrückt gehalten werden muss, haben nicht alle SuS erkannt.
K2b	Beim Bearbeiten der Aufgaben mit der App konnte eine starke Motivation beobachtet werden. Wenn etwas nicht sofort klappte, war leichte Frustration zu erkennen, die von Motivation abgelöst wurde, sobald das Problem gelöst wurde. Alle SuS fanden das Lernen mit der App interessant. 7 SuS stimmten der Aussage voll zu und 3 eher zu.
K2c	Wie das Anwenden der Fakten funktioniert, war nicht allen sofort ersichtlich. So musste die Lehrkraft am Anfang teilweise Hilfestellung geben. Anschließend konnten die Gruppen gut arbeiten. Nach eigener Einschätzung im Anschluss an die Unterrichtsstunde, haben 7 SuS es als einfach (5) oder eher einfach (2) empfunden, Regeln zu erstellen. 3 SuS fanden es eher schwierig. 9 SuS konnten auf dem Fragebogen Fakten mit den Fachbegriffen zum Prisma formulieren.

Id	Zusammenfassung
K2d	Das Aufstellen der Regeln war bei einigen nicht sofort intuitiv zugänglich. Spätestens nach einer Hilfestellung konnte jede Gruppe Regeln erstellen. Die erarbeiteten Wissensbasen zeigen die Ergebnisse (siehe Anhang C.4,C.5,C.6,C.7,C.8). 7 SuS fanden es einfach (6) oder eher einfach (1) Regeln aufzustellen. 3 SuS schätzen diese Aufgabe als eher schwer (2) oder schwer (1) ein. Das erworbene Wissen konnten 5 SuS auf ein neues Problem übertragen und formulierten Regeln für das Prisma im Fragebogen.
K4a	Die prozessbezogene Kompetenz Argumentieren konnte zum Teil während der Erarbeitungsphase beobachtet werden. Dies wurde nicht auf mathematischer Ebene ersichtlich. Die SuS nutzen die zur Verfügung stehenden Körper, um den Partner von einer Meinung zu überzeugen. Die Lehrkraft sah die Kompetenz Argumentieren für die Mathematik nicht im Fokus und kaum vertreten.
K4b	Die prozessbezogene Kompetenz Kommunizieren konnte sehr stark während der Erarbeitungsphase beobachtet werden. Dies zeigt sich an diversen Beispielen in Gesprächen zwischen den Gruppenmitgliedern. Die Lehrkraft gab an, dass der Bereich Kommunizieren in der Schule einen wichtigen Anteil einnimmt und im vorliegenden Fall stark vertreten war und im Vordergrund stand.
K4c	Inhaltsbezogene Kompetenzen in der Mathematik konnten zum Teil angesprochen werden. Die Fachbegriffe Ecken, Kanten und Flächen sowie die Eigenschaften der Körper wurden mehrfach sprachlich wiederholt. Aus Zeitgründen konnten nicht alle SuS jeden Körper bearbeiten. Die Frage 2 des Fragebogen (Ist jeder Würfel ein Quader?) wurde von keinem Schüler richtig begründet.

Tabelle 8.1.: Zusammenfassung der Beschreibung der Beobachtungen und erstellten Artefakte

Neben den Beschreibungen, die konkret den Forschungsfragen zugeordnet werden können, sind einige zusätzliche Beobachtungen (B1, B2, ..., B11) bzw. Ergebnisse im Gesamtkontext und für eine Auswertung interessant.

B1 8 von 10 SuS haben auf die Fragen *Ich würde mir wünschen, von so einer App im Unterricht unterstützt zu werden.* auf dem Fragebogen mit *ja* (7) oder *eher ja* (1) geantwortet. Ein/e weitere/r Schüler/in antwortete *eher nein.* Der/Die Schüler/in der/die Antwort *nein* gegeben hat, fügte als Kommentar an: „Leider noch nicht“ (siehe Fragebogen5 Anhang D.2).

B2 Die Lehrkraft stellt das Spannungsgebiet zwischen der Alltags- und der Fachsprache heraus, die auch in der Grundschule eine wichtige Rolle spiele (vgl. E1 Zeile 12-19).

B3 Die Lehrkraft stellt neben der prozessbezogenen Kompetenz Kommunizieren im Mathematikunterricht den übergeordneten, in der Grundschule wichtigen Aspekt der Kooperation heraus (vgl. E1 Zeile 25-29).

B4 Die Lehrkraft sieht die Einsatzmöglichkeiten solch einer App nicht nur im gezeigten Anwendungsfall. Durch das Sprechen und Formulieren der Sachverhalte könnten fachliche Inhalte geübt und verinnerlicht werden (vgl. E1 Zeile 102-107, Zeile 207-2011).

B5 Die Lehrkraft stellt fest, dass grundsätzlich wenig Vorwissen nötig ist. Fakten und Regeln sollten als Konzepte bekannt sein (vgl. E1 Zeile 138, Zeile 141-142). Ein Schüler ohne Vorwissen kam gut damit zurecht. Ein Schüler sehr schlecht (vgl. E1 Zeile 167-174).

B6 Die Lehrkraft stellt fest, dass die SuS sehr genau sprechen mussten und nicht, wie es im Alltag üblich ist, zu viele Wörter weglassen konnten. Des Weiteren dürfe es nicht zu laut sein. Das sei im Schulalltag eine Herausforderung (vgl. E1 Zeile 157-163).

B7 Die Lehrkraft formuliert mögliche Schwierigkeiten im Bereich der Sprachsensibilität. In stark heterogenen Lerngruppen hinsichtlich der Sprachmöglichkeiten (Probleme, Sätze zu sprechen oder durch Migrationshintergrund Probleme mit der deutschen Sprache) könnte dies problematisch sein. Sie formuliert daraus aber auch eine Chance, solche Probleme gegebenenfalls fördern zu können (vgl. E1 Zeile 175-185).

B8 Die Lehrkraft stellt die Zukunftsbedeutung des Themas heraus. Die Kinder würden jeden Tag mit diesen Medien in Berührung kommen. Somit spiele der Umgang und das Arbeiten mit den Geräten eine immer größere Rolle (vgl. E1 Zeile 213-223). Es sei außerdem ein wichtiges Werkzeug um Lernprozesse zu fördern. Sie betont die Wichtigkeit der Selbstständigkeit und das selbstständige Lernen heraus (vgl. E1 Zeile 295-309).

B9 Die Lehrkraft sieht allgemeine Schwierigkeiten beim Einsatz von Tablets, Smartphones (DT) im Unterricht. So sei die Ausstattung an den Schulen nicht ausreichend vorhanden (vgl. E1 Zeile 285-286). Des Weiteren seien die Lehrer didaktisch und technisch nicht dafür ausgebildet, sodass der Aufwand, den sie in einen Einsatz stecken, kaum im Verhältnis zum Nutzen stehe (vgl. E1 Zeile 290-293). Durch die vielen Inhalte in den Kerncurricula sei es außerdem schwierig etwas unterzubringen, das zusätzlich Zeit koste (vgl. E1 Zeile 278-283).

B10 Die Inhalte der Unterrichtsstunde 2 waren zu umfangreich für die SuS und daher zeitlich nicht alle zu bewältigen. Keine der Gruppen konnte jede der sechs Körper bearbeiten.

B11 Das Arbeitstempo der SuS variierte stark.

8.1.2. Interpretation und Bewertung

Die beschriebenen Ergebnisse und Artefakte aus der Forschung werden im Folgenden bewertet und interpretiert. Dies lässt Aussagen über die einzelnen Forschungsfragen zu, die dadurch beantwortet werden.

Bewertung Frage 1

Die Frage 1 lautet: *Ist es möglich in einer App die Intention durch natürlichsprachliche Eingaben von Kindern der 4. Klasse zu einem fachlichen Thema korrekt zu interpretieren?*. Durch die Einordnung der Beobachtungen und Artefakte zu den beiden Kategorien K1a und K1b wird deutlich, dass es möglich ist, natürlichsprachliche Eingaben von Kindern der 4. Klasse zu einem fachlichen Thema korrekt zu interpretieren.
Auch wenn es in den meisten Fällen keine fehlerhaften Zuordnungen gab, ist trotzdem herauszustellen, dass es teilweise Probleme beim Verstehen der Intention gab. In diesen Fällen ist jedoch ersichtlich, dass die SuS entweder keine Intention hatten (es handelte sich nur um ein Ausprobieren bzw. während des Sprechens wurde abgebrochen) oder das Hineingesprochene nicht zum fachlichen Thema passte. Ein Beispiel konnte identifiziert werden, in dem die Intention und das Gesprochene eindeutig war, die Anwendung dies aber nicht verstanden hat (siehe Anhang C.9 1a). Hier benötigt die Dialogflow-Schnittstelle noch weitere Lerndaten. Die fehlerhafte Zuordnung konnte in diesem Fall korrigiert werden (Trainingsbereich Dialogflow - siehe 5.5.3) und tritt nicht mehr auf.
Das Interpretieren der Entitäten aus dem fachlichen Thema funktionierte überwiegend korrekt. Aber auch hier kam es in wenigen Fällen vor, dass ein Wort nicht zuverlässig erkannt werden konnte. Dies wurde sehr gut bei dem Wort *null* und *nur* ersichtlich, das bei undeutlicher Aussprache nicht korrekt umgesetzt wurde (siehe Anhang C.9 1b)

Bewertung Frage 2

Die Frage 2 lautet: *Kann eine App mit einer HMI ausgestattet werden, die Kindern der 4. Klasse die Interaktion mit einem Prolog-Interpreter ohne syntaktische Hürden er-*

möglicht?. Diese Frage ist vielschichtiger. Für die Beantwortung dieser Frage zeigt diese Arbeit einen potentiellen Ansatz. Die App, die als Einstiegspunkt für eine Interaktion mit einem Prolog-Interpreter dient, wurde mit wenigen Funktionen umgesetzt, um eine Übersichtlichkeit zu gewährleisten. Neben der optischen HMI (GUI Darstellung der Wissensbasis) ist eine weitere, wichtigere Schnittstelle die natürliche Sprache. Lässt diese natürliche Sprache eine Interaktion mit dem Prolog-Interpreter ohne syntaktische Hürden zu? Nein, syntaktische Hürden gibt es auch hier. Die SuS mussten lernen, was Fakten und Regeln sind. Vor allem bei Regeln wird die *Wenn ..., dann...*-Beziehung forciert. Ein weiteres Argument zeigt noch einmal genauer, dass es wahrscheinlich immer, aber vor allem bei natürlicher Sprache, eine syntaktische Hürde gibt. Syntaktik in Programmiersprachen folgt einer Grammatik der entsprechenden Sprache. Diese ist ebenfalls in der natürlichen Sprache vorhanden. Daraus folgt, dass SuS, die mit der Grammatik der natürlichen Sprache Probleme haben, vor höheren syntaktischen Hürden beim Bedienen der App stehen, als andere. Dieses Argument wird in Abschnitt 8.1 unter Beobachtung 7 (B7) beschrieben.
Es ist somit nur relevant, wie hoch diese syntaktischen Hürden sind und nicht, ob diese existieren. Ein Ziel der Forschung ist es, herauszufinden, wie gut SuS der 4. Klasse mit dem Prolog-Interpreter interagieren können.
Die Befragung durch die Fragebogen hat ergeben, dass das allgemeine Bedienen der App bei fast allen SuS problemlos funktionierte. Einer der Schüler, der kein Vorwissen aus der Unterrichtsstunde davor hatte, kam nicht gut mit der App zurecht. Hier könnte eine Kombination aus fehlender Motivation durch Frustration (fehlendes Vorwissen) und eher geringerem Leistungsvermögen den Ausschlag gegeben haben. Hilfestellungen wurden beim Löschen von Fakten und Regeln aus der Wissensbasis gegeben. Die Funktionsweise des Mikrophone-Buttons wurde teilweise anders interpretiert. Der Assistent ist auf die übliche Weise implementiert, das heißt, durch Antippen fängt das Gerät an zu lauschen und beendet automatisch die Aufnahme, sobald der Nutzer den Satz beendet. Durch Nachrichten-Apps wie zum Beispiel WhatsApp könnte eine andere Funktionsweise bekannt sein. Diese sieht vor, den Button so lange gedrückt zu halten, wie etwas aufgenommen werden soll.
Die Motivation beim Bearbeiten der App war sehr hoch. Dies spiegelt sich in den Fragebogen und in den Einschätzungen der Lehrkraft wider.
Die Voraussetzungen für eine Interaktion mit dem Prolog-Interpreter (ausreichend klare Bedienung der Entwicklungsumgebung und Motivation) sind gegeben. Wie hoch waren im Forschungsrahmen die syntaktischen Hürden? Während der Bearbeitung konnte beobachtet werden, dass die meisten Bearbeitungsschritte ohne Hilfe stattfanden. Sowohl beim Erstellen von Regeln als auch von Fakten. Dies spiegelt auch die Einschätzung der

SuS wider. Wenn es Probleme gab, konnten diese anfangs durch Tipps der Lehrenden gelöst werden. Der Fragebogen enthält darüber hinaus Fragen zu Fakten und Regeln eines nicht bekannten Körpers. Ziel war es zu untersuchen, ob die SuS die Konzepte hier ebenfalls anwenden können. Dies wurde zum Teil gut umgesetzt.
Zusammenfassend kann die Frage insoweit beantwortet werden, dass die erstellte Anwendung eine einfache Interaktion mit einem Prolog-Interpreter ermöglicht, indem die syntaktischen Hürden durch das Verwenden natürlicher Sprache herabgesetzt werden. Dies bestätigt die Lehrkraft ebenfalls mit der Aussage des geringen Vorwissens, das für eine Verwendung nötig ist (vgl. Beobachtung 5 (B5)).

Bewertung Frage 4

Die Frage 4 lautet: *Hilft die App mit logischem Programmierparadigma und im erarbeiteten Unterrichtsrahmen Kindern der 4. Klasse, fachliche Zusammenhänge sprachlich zu formulieren? Werden die gewünschten Kompetenzbereiche angesprochen?* Die gewünschten Kompetenzbereiche können in prozess- und inhaltsbezogene Kompetenzbereiche unterteilt werden. Als prozessbezogene Kompetenzbereiche sollen das Argumentieren und das Kommunizieren im Vordergrund stehen. Inhaltsbezogene Kompetenzen orientieren sich an dem Bereich Körper in der Geometrie. Bezüglich der Kompetenz Argumentieren lässt sich feststellen, dass die SuS kaum über fachliche Aspekte diskutiert oder diese begründet haben. Lediglich die realen Körper als Hilfsmittel wurden genutzt, um die Gruppenpartner zu überzeugen. Die Lehrkraft spiegelt dies im Interview wider. Das Kommunizieren hingegen konnte als vordergründige Kompetenz identifiziert werden. Dies zeigen die Beobachtungen genauso wie die Einschätzungen der Lehrkraft. Darüber hinaus wurde noch eine nicht fachspezifische Kompetenz angesprochen. Aus pädagogischer Sicht sei in der Schule der soziale Aspekt kontinuierlich zu fördern. Dies rückte im durchgeführten Rahmen ebenfalls stark in den Vordergrund.
Hinsichtlich der inhaltsbezogenen Kompetenzen konnte verstärkt auf die Fachbegriffe und Eigenschaften der Körper eingegangen werden. Die erste Frage des Fragebogens haben 9 von 10 SuS richtig beantwortet. Der Fokus sollte allerdings nicht lediglich auf den fachlichen Fakten liegen, sondern explizit auf Zusammenhängen und Unterschieden der Körper. Hier zeigt die Beantwortung in Frage zwei, dass die Unterschiede beim Würfel und Quader nicht ausreichend begründet wurden. Das Bearbeiten der einzelnen Aufgaben hat mehr Zeit in Anspruch genommen als ursprünglich geplant. Dadurch konnte keine Gruppe alle Körper bearbeiten. Somit fällt die bewusst gewollte Differenzierung zwischen dem Quader und dem Würfel heraus, was dieses Resultat begründen

kann. Außerdem wurde die fachliche Seite des Fragebogens zum Teil nicht ausreichend konzentriert und gewissenhaft bearbeitet. Zum einen war den SuS bekannt, dass keine Bewertung erfolgen würde und zum anderen war die Stunde während der Bearbeitung des Fragebogens zeitlich beendet, sodass die Motivation zum gewissenhaften Ausfüllen des Fragebogens bei einigen SuS gering war.
Durch die gegebenen Beobachtungen und Ergebnisse ergibt sich der Schluss, dass die App den SuS im erarbeiteten Unterrichtsrahmen hilft, fachliche Zusammenhänge sprachlich zu formulieren. Dies wird mit Hilfe deklarativer Programmierung erreicht, die sich am logischen Paradigma anlehnt. Der direkte Bezug zu logischer Programmierung wird den Kindern nicht bewusst, da dieser abstrahiert wird. Die gewünschten Kompetenzen werden zum Teil angesprochen. Kommunizieren nimmt eine zentrale Position ein, wohingegen das Argumentieren eher vernachlässigt wird. Inhalte werden vertieft, Unterschiede sind jedoch nicht stark fokussiert, obwohl sie durch die Kinder implizit formuliert werden.

8.2. Fallstudien

Die 3 Frage lautet: *Wie kann die App intern strukturiert sein, um zukünftige Fortschritte bei Frameworks und Technologien zur einfachen Benutzerinteraktion ohne hohe Änderungsaufwände in der App nutzen zu können?* Diese soll mit Hilfe von potentiellen Anpassungen an der App argumentiert werden. Als Grundlage dienen der bestehende App-Quellcode, die *Fallstudien* und argumentative Beobachtungen. Zum Bewerten sind die Bewertungskriterien (siehe Abschnitt 7.4) heranzuziehen. Bei der Bewertung und Argumentation steht der Aspekt Wartbarkeit der Software-Produktqualität (Maintainability) mit deren Unterkategorien im Mittelpunkt (vgl. Abschnitt 2.4). Die Fallstudien sind in Abschnitt 7.3 zu finden. Die Anhänge C.10, C.11 und C.12 zeigen die Anwendung der Fallstudien. Eine Beschreibung und Einordnung in die Bewertungskriterien ist im Anhang C.13 verfügbar.

8.2.1. Beschreibung

Fallstudie 1
Durch die Rückmeldungen und Beobachtungen (vgl. Abschnitt 8.1.2) wurde deutlich, dass einige Schülerinnen und Schüler mit der Implementierung des Aufnahme-Buttons

Probleme hatten. Dies zeigt die Sinnhaftigkeit solch einer Anpassung. Um diese Anforderung umsetzen zu können, müssen in jedem Modul Änderungen implementiert werden. Da ein neuer Service entstehen soll, erhält das domain-Modul ein Interface, damit ein UseCase diesen steuern kann. Die steuernden UseCases werden ebenfalls hinzugefügt. Das data-Modul erhält einen neuen Service, der die RecognitionService-Schnittstelle implementiert. Die Funktionalität dafür wird aus dem *ApiAiServiceHandler* entfernt. Die View (app-Modul) muss eine Möglichkeit erhalten, nicht lediglich beim Drücken das Event auszulösen, sondern beim Drücken das startende und beim Loslassen das stoppende Event.
Anzumerken ist, dass wenige Zeilen Quellcode geändert werden müssen. Hauptsächlich wird Funktionalität durch neue Klassen und Interfaces hinzugefügt. Des Weiteren ist eindeutig, welche Klassen anzupassen sind und in welchen Fällen neue hinzukommen.

Fallstudie 2
Die Service-Klasse, die für die Kommunikation mit dem NLP-Dienst zuständig ist, wird ausgetauscht. Diese muss dann lediglich das korrekte Interface aus dem domain-Modul implementieren (*NLPHandler* ehemals *APIAIHandler*). Im domain-Modul müssen die Klassen angepasst werden, die für den Aufruf der Schnittstelle zuständig sind. Außerdem interpretiert ein UseCase die Antwort. Diese beiden Berührungspunkte müssen überprüft und gegebenenfalls angepasst werden. Eine Änderung im app-Modul ist nicht nötig. Auf die View hat solch ein Wechsel auf Grund der Abhängigkeitsregeln keinen Einfluss.
Hier wird deutlich, dass eine Trennung nach der softwareseitigen Abhängigkeit (lose Kopplung - Modularität) prinzipiell gut umgesetzt wurde, jedoch durch die Namensgebung und Fokussierung auf Dialogflow keine saubere Trennung vollzogen wurde. Es wäre abzuwägen, ob das data-Modul Auswertungen der spezifischen Response-Objekte eines externen Dienstes übernehmen soll. Falls ja, wäre eine komplette Trennung möglich. Allerdings wird Logik, die teilweise als geschäftsrelevant bezeichnet werden kann, im data-Modul ausgeführt. Die derzeitige Lösung überlässt das Interpretieren der Antwort dem entsprechenden UseCase im domain-Modul.

Fallstudie 3
Um die Darstellung der Wissensbasis anzupassen, sind zwei Hauptarbeitsbereiche identifizierbar. Die ViewModel-Objekte sind für die Anzeige und die Optik der Daten zuständig und müssen daher angepasst und erweitert werden. Des Weiteren ist die Schnittstelle zur Logik ein Mapper, der die Model-Objekte in ViewModel-Objekte überführt und umgekehrt. Dieser muss ebenfalls an die neuen ViewModel-Objekte angepasst werden. Durch die klare Trennung der View (app-Modul) zur Businesslogik (domain-Modul) sind Änderungen lediglich im app-Modul zu tätigen. Dank der Mapper in den verschiedenen

Modulen gibt es keine weitere Abhängigkeit. Somit hat solch eine Anpassung keinen Einfluss auf das data-Modul und die Persistierung der anzuzeigenden Elemente.

8.2.2. Interpretation und Bewertung

Die Software ist modular aufgebaut. Es existieren auf oberster Ebene drei Module. Im Quellcode wurden mit Hilfe eines Frameworks darüber hinaus noch einzelne Module nach ihrer Funktion unterteilt. Es wäre möglich, die App noch modularer aufzubauen. Dies würde die Komplexität allerdings stark erhöhen. Ob dies im Verhältnis zum Nutzen wäre, lässt sich anzweifeln. Die Fallstudien zeigen eindrucksvoll, wie auf Änderungen innerhalb der App reagiert werden kann. Hier kommen das Single Responsibility Principle und das Open Closed Principle (vgl. Abschnitt 2.3.1) stark zur Geltung. Das MVP-Pattern für die View erweist sich vor allem in den Fallstudien 1 und 3 als besonders wertvoll. Durch die Trennung der View und des Presenters können Änderungen ohne hohen Aufwand implementiert werden.

Android-Apps sind durch ihre Fokussierung auf View-Interaktion schwer automatisiert testbar. Businesslogik, die sich in Apps befindet, sollte möglichst ohne Abhängigkeiten zu Frameworks implementiert sein. Dies wird durch das domain-Modul realisiert. Hier kann die Businesslogik effizient getestet werden. View-Tests sind nur mit erheblichem Aufwand möglich, von dem hier abgesehen wurde. Die interne App-Struktur begünstigt demnach das Testen der Businesslogik, kann allerdings nicht die allgemeinen plattformspezifischen Probleme des Testens lösen.

Aus den Bewertungskriterien lässt sich somit hinsichtlich der Fragestellung ableiten, dass die angewendete App-Architektur gut für Software geeignet ist, die auf Wartbarkeit Wert legt. Dies liegt vor allem am Trennen der Module und Einhalten der Abhängigkeitsregeln. Clean Architecture ist hierbei kein notwendiges Kriterium, hilft aber dabei, die Design Prinzipien (SOLID siehe Abschnitt 2.3.1, DDD siehe Abschnitt 2.3.2) anzuwenden und die Umsetzung zu standardisieren. Abschnitt 5.4 beschreibt die konkrete Umsetzung von Clean Architecture. Im Vergleich zu deren Zielen (siehe Abschnitt 2.3.4) ergibt sich, dass ebendiese erreicht wurden. Sie decken sich darüber hinaus zum Teil mit den drei aufgestellten Bewertungskriterien. Somit wird deutlich, dass die Architekturwahl zielführend zur Fragestellung 3 war.

Ein weiterer wichtiger Aspekt ist die Wiedererkennbarkeit eines Architektur-Patterns. Ein Entwickler, der mit dem Clean Architecture Ansatz vertraut ist, kann die App ohne Probleme warten oder erweitern.

8.3. Reflexion

Dieser Abschnitt reflektiert die erarbeiteten Unterrichtsstunden, die Erhebungsmethode, die Fallstudien und die Ergebnisse.

Die erstellte App wurde mit zwei Unterrichtsstunden in einer qualitativen Sozialforschung mit Kindern der 4. Klasse an einem fachlichen Thema erprobt. Die erste Unterrichtsstunde hatte als Ziel, den SuS das Konzept zu Regeln und Fakten beizubringen. Dies sollte sie befähigen mit der App zu arbeiten. Durch die Alltagsbedeutung der Fakten und Regeln hatten die SuS einen guten Zugang zum Thema. Die Arbeit mit dem Memory und das Kennenlernen einer Wissensbasis anhand eines Seils ist bei vielen geglückt. Die abstrakten Konzepte der Wissensbasis und wie damit programmiert wird, waren jedoch nicht allen Schülern zugänglich. Hier hätte eine Präsentation der App gegebenenfalls genutzt werden können, um die Motivation bei allen Kindern zu stärken. Der zeitliche Ablauf entsprach der Planung in der Lehrskizze. Alle geplanten Inhalte und Konzepte konnten mit den beschriebenen Methoden durchgeführt werden. Die zweite Unterrichtsstunde startete mit der angesprochenen Motivation. Durch direktes Einbeziehen der SuS beim Konzipieren der Wissensbasis wurde das Verlangen geweckt, dies selbstständig auszuprobieren. Das Aktivieren des Vorwissens zu den Fachbegriffen wurde durch das Plakat unterstützt. In der anschließenden Arbeitsphase konnten die SuS in Gruppen selbstständig ihre Wissensbasis aufbauen. Durch eine Fehleinschätzung bezüglich der Bearbeitungsdauer (siehe Abschnitt 8.1.1 B10) konnte keine der Gruppen alle Körper hinzufügen.

Die Erhebung wurde mit Hilfe von Fragebogen, einem Interviewleitfaden und Fallstudien zur App-Erweiterung durchgeführt. Die fachlichen Fragen zu Körpern und Zusammenhängen waren nicht ideal gewählt. So konnte eine grundsätzliche Tendenz abgeleitet, jedoch nicht signifikant gute Antworten hervorgebracht werden. Weiterhin wurde das Prisma als neu zu beschreibender Körper gewählt. Ziel war es zu untersuchen, wie die SuS die Fragen beantworten. Konnten die erlernten Konzepte aus der App umgesetzt werden? Aus mathematisch fachlicher Sicht ist die Wahl des Prisma diskussionswürdig, da eine Abgrenzung zum Quader oder Kegel für die Kinder mit dem vorhandenen Vorwissen nicht möglich ist. Die Art der Beantwortung war jedoch für die Auswertung hilfreich. Aus mathematischer Sicht kann reflektiert werden, dass die fachlichen Fragen des Fragebogens spezifischer hinsichtlich des Verständnisses von Zusammenhängen hätten formuliert werden können. Die Fragen hinsichtlich der Bedienung der App konnten von den SuS gut beantwortet werden und waren zielführend. Durch die Zeitproblematik füllten nicht alle SuS die Fragebogen (vor allem im fachlichen Bereich) mit großer

Zielstrebigkeit und Konzentration aus. Hier müsste auf eine ruhige Umgebung ohne großen Zeitdruck geachtet werden. Das Interview mit der Lehrperson war aufschlussreich und zielführend. Das geplante zweite Interview zu den Ergebnissen der Fragebogen der SuS entfiel, da keine Notwendigkeit dafür bestand. Die Leitfragen des Interviews waren sinnvoll gewählt. Neben den thematisch gewünschten Einschätzungen waren einige Ausführungen teilweise etwas zu weit von den konkreten Zielen der Arbeit entfernt. So wurden neben den gewollten Einschätzungen sehr interessante, aber nicht konkret in dieser Arbeit auswertbare Antworten gegeben. Um die technische Architekturfrage beantworten zu können, sollten Fallstudien helfen, konkrete Einschätzungen begründen zu können.

Das Auswerten der aufgestellten Fallstudien und Zuordnen zu den Bewertungskategorien war nicht so eindeutig möglich wie bei der umgesetzten qualitativen Sozialforschung. Die einzelnen Kategorien und Prinzipien bedingen sich teilweise stark. Es war jedoch möglich zu zeigen, dass der implementierte Architekturansatz viele Vorteile hat. Die kritische Auseinandersetzung mit dieser Architektur ist im Rahmen der Arbeit nicht vollumfänglich möglich. Jedoch ist es sinnvoll an dieser Stelle, einige kritische Argumente einmal hervorzuheben. Eine Anmerkung kann die Menge an Quellcode sein, der zum Schreiben der Anwendung entstanden ist. Dies hat sich jedoch bereits durch den Einsatz von Kotlin reduziert, da diese Sprache expressiver im Vergleich zu Java ist. Als weiterer Kritikpunkt kann angeführt werden, dass sich eine App im Vergleich zu beispielsweise Java Enterprise Anwendungen durch eine eher geringe Lebensdauer auszeichnet. Durch einen verhältnismäßig geringen Gesamtrahmen und ein niedriges Auftragsvolumen lohne sich eine solche Architektur nicht zwangsläufig. Des Weiteren würde die eher geringe Anzahl an Entwicklern und deren geringe Fluktuation in den Projekten gegen einen initialen Zusatzaufwand für App-Projekte sprechen. Dem würde ich dahingehend widersprechen, dass eine Architektur immer am jeweiligen Projekt und dessen Anforderungen gemessen werden sollte. Ein Prototyp, der nicht gewartet oder weiterentwickelt wird, benötigt nicht zwangsläufig eine solche Architektur. Bei allen Projekten, die produktiv eingesetzt werden und in Wartung übergehen, ist die gezeigte Architektur eine als sinnvoll zu beachtende Option.

Die Auswertung der Ergebnisse hat gezeigt, dass eine App entstanden ist, die die SuS motiviert und die potentiell in einem bestehenden Unterrichtsrahmen einsetzbar sein könnte. Hier müsste für eine ruhige oder weiträumig verteilte Lernumgebung gesorgt werden. Positiv ist hervorzuheben, dass es wenig Vorwissen bedarf (siehe Abschnitt 8.1.1 B5). Die App wurde von den SuS als Bereicherung empfunden und sie kamen größtenteils mit ihrer Bedienung gut zurecht. Somit entstand ein System, das die In-

tention beim deklarativen Programmieren mit natürlicher Sprache versteht. Durch die in dieser Arbeit starke Fokussierung auf ein fachliches Thema (geometrische Körper) konnte der Einsatz im Mathematikunterricht gezeigt werden. Es lässt sich prinzipiell aber auch auf andere Themen übertragen (siehe Abschnitt 8.1.1 B4). Hinsichtlich der Kompetenzen fördert die App im durchgeführten Unterrichtsrahmen nicht wie erwartet das Argumentieren. Das Kommunizieren und der soziale Aspekt (siehe Abschnitt 8.1.1 B3) stehen im Vordergrund. Bereits das Verwenden der Tablets ist für die SuS ein wichtiger Lernprozess, der hinsichtlich der Gegenwarts- und Zukunftsbedeutung bedeutsamer Bestandteil des Unterrichts sein sollte (siehe Abschnitt 8.1.1 B8).

Dem Thema *Sprache* fällt in der (Grund-)Schule eine besondere, im Vorfeld nicht explizit formulierte Gewichtung zu. Es existiert das Spannungsfeld zwischen der Alltags- und der Fachsprache (siehe Abschnitt 8.1.1 B2), das durch die App fokussiert wird (Benennen der Fachbegriffe). Sprachlich mussten die SuS sehr genau auf die Aussprache achten (siehe Abschnitt 8.1.1 B6). Dies eröffnet nicht beachtete Chancen und Risiken. Inklusionsschüler/innen mit sprachlichen Beeinträchtigungen hätten ebenso Schwierigkeiten mit der App wie Kinder mit Migrationshintergrund, die die deutsche Sprache noch nicht ausreichend beherrschen. Dies könne zwar als Chance gesehen werden, das korrekte Formulieren zu üben, jedoch ist die App auf solch einen Anwendungsfall hin nicht optimiert.

9. Schluss

In diesem Kapitel wird in einem Fazit die Arbeit bewertet. Diese Schlussfolgerungen zeigen die Bedeutung der Ergebnisse auf. Im letzten Abschnitt gibt ein Ausblick einen Überblick über mögliche nächste Schritte, die auf Grundlage dieser Arbeit sinnvoll erscheinen.

9.1. Fazit

Im Rahmen der Arbeit entstand eine Android-Anwendung, die eine Entwicklungsumgebung für logische Programmierung darstellt. Als Benutzerschnittstelle wurde neben der grafischen Darstellung die natürliche Sprache gewählt, um syntaktische Hürden herabzusetzen, die bei der Bedienung eines Prolog-Interpreters auftreten. Hierbei wurde nur ein Teil aus der logischen Programmierung umgesetzt und in den Kontext einer Unterrichtsstunde für SuS der 4. Klasse gestellt.
Meiner Meinung nach zeigt diese Arbeit den Inbegriff der angewandten Informatik. Es wurde ein Informatikthema (NLP und digitale Assistenten) genutzt und in einen angewandten Forschungskontext gestellt. Somit konnte ein Ansatz der natürlichsprachlichen Programmierung in einem Feldversuch in der Grundschule durchgeführt werden. Diese Forschungsarbeit verbindet die technischen Informatikthemen mit didaktischen Überlegungen für einen sinnvollen Einsatz der Technologien. Der Fokus während dieser Ausarbeitung verschob sich von Prolog hin auf eine natürlichsprachliche deklarative Möglichkeit der Programmierung. Prolog wurde zum Träger und Werkzeug und ist nicht mehr direkter Forschungsgegenstand. Diese Entwicklung war meiner Einschätzung nach wichtig und richtig. Durch den bisher gezeigten Anwendungsfall ist das entwickelte Gesamtsystem noch eine domänenspezifische Programmiersprache. Auf dem Weg hin zu einer vollwertigen natürlichen Programmierung müssten die Möglichkeiten allgemeingültiger werden.

Somit ist die natürliche Sprache ein wesentlicher Faktor und innerhalb der Arbeit entscheidender als das logische Programmierparadigma. Die SuS haben Grundlagen zu Fakten und Regeln erfahren sowie den Einsatz eines Expertensystems. Dieses deklarativ zu beschreibende und zu programmierende System kann, muss jedoch nicht mit Hilfe von logischer Programmierung umgesetzt sein. Andere regelbasierte Systeme könnten ebenso als Backend eingesetzt werden. Dank der Architekturentscheidungen ist eine Anpassung mit verhältnismäßig wenig Aufwand möglich.
Ein Ziel der Arbeit war das Interagieren mit einem Prolog-Interpreter ohne syntaktische Hürden. Bei der Bewertung dieser Frage (siehe Abschnitt 8.1.2) wurde bereits dargelegt, dass eine Programmierung *ohne* syntaktische Hürden nicht möglich ist. Auch wenn deutlich wird, dass geringe syntaktische Hürden im Fokus sind, wurde die Bedeutung der natürlichen Sprache als Hürde nicht beachtet. Dieser Fokus muss hinsichtlich aller NLP-Systeme bedacht werden.

9.2. Ausblick

Aus didaktischer Sicht wäre es sinnvoll, zu erforschen, ob eine Integration in den bestehenden Unterricht möglich wäre. Da der untersuchte Anwendungsfall argumentativ beschreibt, dass eine Integration gut möglich sei, die Durchführung aber isoliert geschehen ist, könnte eine konkrete Integration erforscht werden. Ein weiterer Schritt ist die Erarbeitung zusätzlicher, sinnvoller Themen, die mit Hilfe der App im Unterricht behandelt werden könnten. Das NLP-Backend müsste dementsprechend trainiert werden. Hinsichtlich der App ist interessant, inwiefern sich Anforderungen ergeben, um stärker zwischen den heterogenen Sprachmöglichkeiten der SuS zu differenzieren oder ob dies durch eine gewählte Methodik ausgeglichen werden kann bzw. soll.

Die Dialogflow-Schnittstelle (D) ist für eine Kommunikation mit dem Nutzer prädestiniert. Somit könnte ein Dialog den Lernenden (L) unterstützen. Beispiel:
L: Ich möchte programmieren. *D: Erstelle doch als erstes einen Fakt. Sag mir, was du siehst.*

Eine Erweiterung könnte sein, Fakten in der App nicht vollständig selbst einzufügen, sondern durch externe Sensoren melden zu lassen. Als Beispiel sind Bewegungsmelder, Kontakt- oder Wärmesensor zu nennen, die Werte an die App liefern. Die App könnte diese Werte als Fakten darstellen. Die Schülerinnen und Schüler haben mit den Regeln die Möglichkeit, verschiedene Systeme zu konzipieren (Warnsysteme bei Wärme, offene Fenster, Bewegungen etc.) und als Regelsystem zu programmieren.

Hinsichtlich des Umfangs von Prolog wäre eine Vervollständigung der Sprachmittel denkbar. Derzeit sind Konzepte wie Listen und Variablen nicht mit umgesetzt.
Ein Prolog-Interpreter ist nicht die einzige Möglichkeit, um das Expertensystem aus Regeln und Fakten auszuwerten. Eine Überlegung könnte sein, andere regelbasierte Programmiersprachen oder domänen-spezifische Sprachen (DSL) zu integrieren, um spezielle Anforderungen zu erfüllen.

Das verwendete NLP-Backend Dialogflow kann ausgetauscht werden. Derzeit benötigt die App eine dauerhafte Internetverbindung, um mit diesem Backend zu kommunizieren. Aktuelle Smartphones und Tablets erhalten separate Chips, die explizit zum Ausführen von Maschine-Learning-Algorithmen optimiert sind (NLP verwendet Machine-Learning-Algorithmen). Damit eröffnet sich die Möglichkeit, den Drittdienst mit Internetverbindung abzulösen und eine interne NLP-Lösung zu entwickeln.

Der wohl abstrakteste, aber nicht zu vernachlässigende Gedanke zum natürlichsprachlichen, deklarativen Programmieren ist die gesellschaftliche und wirtschaftliche Bedeutung. Arbeitsbereiche digitalisieren sich zunehmend, Berufszweige fallen komplett weg und Informatiker werden benötigt. Natürlichsprachliche, deklarative Programmierung kann eine Chance sein, dass nicht ausgebildete Informatiker in ihrer Fachdomäne Maschinen und Prozesse nicht nur steuern, sondern auch programmieren können, ohne über das IT-Wissen zu verfügen. So kann eine Abstraktionsebene zum Programmieren geschaffen werden, die durch den aktuell wachsenden Maschine-Learning und NLP Bereich in den nächsten Jahren an Bedeutung gewinnen könnte. Eine Analyse verwandter Arbeiten, Konzepte und Lösungsansätze wäre demnach genauso sinnvoll wie eine Integration in solch einem Kontext.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Glossar

API	application programming interface - Programmierschnittstelle.
Boilerplate Code	Programmcode, der nicht für die Funktionalität (Businesslogik) geschrieben ist, sondern meist architekturspezifische Funktionen übernimmt.
Broadcast	Senden einer Nachricht an alle (potentiellen) Empfänger, ohne diese konkret anzusprechen..
Bytecode	Virtuelle Maschinen nutzen diesen als Vorstufe zum Maschinencode.
Callbacks	Funktion um Daten einer Funktion/Bibliothek nach Beendigung des definierten Anwendungsfalls zur aufrufenden Funktion zurückzugeben.
Google I/O	Entwicklerkonferenz von Google, auf der jährlich die Neuheiten präsentiert werden.
GUI	graphical user interface - Grafische Benutzerschnittstelle, an der der Nutzer seine Eingaben zum System tätigt.
HMI	human–machine interface - Benutzerschnittstelle (Beispiel: Spracheingabe, GUI).
IDE	integrated development environment - integrierte Entwicklungsumgebung.
JVM	Java Virtual Machine.

Kotlin	Programmiersprache - entwickelt von JetBrains.
Linux	Mehrbenutzer-Betriebssystem, das auf dem freien Linux-Kernel aufsetzt. Initiiert durch Linus Torvald.
Literal	Atomare Aussage oder deren Negation - positive Literale und negative Literale.
MVC	Model View Controller - Software Design Pattern für Oberflächenanwendungen.
paradigma	Grundsätzlicher Ansatz Computerberechnungen auszudrücken. Unabhängig von der Syntax der konkreten Sprache.
Scala	Funktionale (nicht rein-funktional) und objektorientierte Programmiersprache auf der JVM.
SuS	Schülerinnen und Schüler.
Unifikation	Vereinheitlichung prädikatenlogischer Ausdrücke durch ersetzen der Variablen durch geeignete Terme.
Unix	Proprietäres Mehrbenutzer-Betriebssystem.
XML	Extensible Markup Language - hierarchisch strukturierter Auszeichnungssprache im Textformat.

Literatur

Android-Developer (2017a). *Android API Guide - Intents and Intent Filters*. URL: https://developer.android.com/guide/components/intents-filters.html (besucht am 04. 12. 2017).

— (2017b). *Android API Referenz - Activity*. URL: https://developer.android.com/reference/android/app/Activity.html (besucht am 16. 11. 2017).

Backfield, Joshua (2014). *Becoming functional*. Sebastopol, CA: O'Reilly Media. ISBN: 978-1-449-36817-3.

Ballard, Bruce W. und Alan W. Biermann (1979). "Programming in natural language". In: *Proceedings of the 1979 annual conference on - ACM 79*. ACM Press.

Bogner, A., B. Littig und W. Menz (2013). *Das Experteninterview: Theorie, Methode, Anwendung*. VS Verlag für Sozialwissenschaften. ISBN: 9783322932709.

Brüsemeister, Thomas (2008). *Qualitative Forschung: Ein Überblick*. Studientexte zur Soziologie. VS Verlag für Sozialwissenschaften. ISBN: 9783531911823.

Carkci, Matt (2014). *Dataflow and reactive programming systems a practical guide*. S.l.S.l: Matt CarkciMatt Carkci. ISBN: 9781497422445.

Cervone, Francesco (2017). *Model-View-Presenter: Android guidelines*. URL: https://medium.com/@cervonefrancesco/model-view-presenter-android-guidelines-94970b430ddf (besucht am 03. 12. 2017).

Chiusano, Paul (2014). *Functional programming in Scala*. Shelter Island, NY: Manning Publications. ISBN: 978-1617290657.

Cohn, M. (2004). *User Stories Applied: For Agile Software Development*. Addison-Wesley signature series. Addison-Wesley. ISBN: 9780321205681.

Dale, R., H. Moisl und H. Somers (2000). *Handbook of Natural Language Processing*. Taylor & Francis. ISBN: 9780824790004.

Evans, Eric (2004). *Domain-driven design : tackling complexity in the heart of software*. Boston: Addison-Wesley. ISBN: 978-0321125217.

Flick, Uwe (2011). *Qualitative Sozialforschung : eine Einführung*. Reinbek bei Hamburg: Rowohlt-Taschenbuch-Verl. ISBN: 978-3-499-55694-4.

Foegen, Malte et al. (2014). *Der Ultimative Scrum Guide 2.0*. Darmstadt: wibas GmbH. ISBN: 978-3-981-58375-5.

Ford, Neal (2014). *Functional thinking*. Sebastopol, CA: O'Reilly Media. ISBN: 978-1-449-36551-6.

Fuß, S. und U. Karbach (2014). *Grundlagen der Transkription: Eine praktische Einführung*. Uni-Taschenbücher. UTB GmbH. ISBN: 9783825241858.

Gabbrielli, Maurizio und Simone Martini (2010). *Programming languages : principles and paradigms*. London New York: Springer. ISBN: 978-1-84882-913-8.

Gage, N.L., D.C. Berliner und G. Bach (1996). *Pädagogische Psychologie*. U-&-S-Pädagogik. Beltz. ISBN: 9783621273114.

Gesellschaft für Informatik (GI) e. V. (2017). *Kompetenzen für informatische Bildung im Primarbereich*. URL: http://bscw.ham.nw.schule.de/pub/bscw.cgi/d5795069/LP_IF.pdf (besucht am 05. 12. 2017).

Gharbi, M. et al. (2017). *Basiswissen fór Softwarearchitekten: Aus- und Weiterbildung nach iSAQB-Standard zum Certified Professional for Software Architecture – Foundation Level*. dpunkt.verlag. ISBN: 9783960883333.

Ghosh, Debasish (2017). *Functional and reactive domain modeling*. Shelter Island, NY: Manning Publications Co. ISBN: 978–1-61729-224-8.

Google (2017a). *Material Design - Components - Buttons: Floating Action Button*. URL: https://material.io/guidelines/components/buttons-floating-action-button.html (besucht am 04. 12. 2017).

— (2017b). *Material Design - Components - Cards*. URL: hhttps://material.io/guidelines/components/cards.html (besucht am 04. 12. 2017).

— (2017c). *Material Design - Patterns - Errors*. URL: https://material.io/guidelines/patterns/errors.html (besucht am 04. 12. 2017).

— (2017d). *Material Design - Platforms - Platform adaptation*. URL: https://material.io/guidelines/platforms/platform-adaptation.html (besucht am 04. 12. 2017).

Hartmann, W., M. Näf und R. Reichert (2007). *Informatikunterricht planen und durchführen*. eXamen.press. Springer Berlin Heidelberg. ISBN: 9783540344858.

Hattermann, Mathias et al. (2015). "Geometrie: Leitidee Raum und Form". In: *Handbuch der Mathematikdidaktik*. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 185–219.

Haywood, Dan (2009). *Domain-driven design using naked objects*. Raleigh, N.C: Pragmatic Bookshelf. ISBN: 978-1-93435-644-9.

Helfferich, Cornelia (2014). "Leitfaden- und Experteninterviews". In: *Handbuch Methoden der empirischen Sozialforschung*. Hrsg. von Nina Baur und Jörg Blasius. Wiesbaden: Springer Fachmedien Wiesbaden, S. 559–574. ISBN: 978-3-531-18939-0.

Hoffmann, D.W. (2008). *Software-Qualität*. EXamen. press Series. Springer Berlin Heidelberg. ISBN: 9783540763239.

Hubwieser, P. (2007). *Didaktik der Informatik: Grundlagen, Konzepte, Beispiele*. eXamen.press. Springer Berlin Heidelberg. ISBN: 9783540724780.

ISO/IEC (2011). *ISO/IEC 25010: Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Software and quality in use models*. Standard. Geneva, CH: International Organization for Standardization.

Jacobson, Ivar (1992). *Object-oriented software engineering : a use case driven approach*. New York Wokingham, Eng. Reading, Mass: ACM Press Addison-Wesley Pub. ISBN: 0-201-54435-0.

Jörissen, Stefan und Barbara Schmidt-Thieme (2015). "Darstellen und Kommunizieren". In: *Handbuch der Mathematikdidaktik*. Hrsg. von Regina Bruder et al. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 385–408.

Kallus, Konrad (2016). *Erstellung von Fragebogen*. Wien: facultas. ISBN: 978-3-8252-4465-1.

Kleine Büning, Hans und Stefan Schmitgen (1988). *PROLOG : Grundlagen und Anwendungen : mit zahlreichen Abbildungen, Tabellen und Programmbeispielen*. Stuttgart: Teubner. ISBN: 3-519-12484-X.

KMK (2003). *Bildungsstandards im Fach Mathematik für den Mittleren Schulabschluss*. URL: http://www.kmk.org/fileadmin/Dateien/veroeffentlichungen_beschluesse/2003/2003_12_04-Bildungsstandards-Mathe-Mittleren-SA.pdf (besucht am 14. 11. 2017).

— (2004). *Bildungsstandards im Fach Mathematik für den Primarbereich*. URL: https://www.kmk.org/fileadmin/Dateien/veroeffentlichungen_beschluesse/2004/2004_10_15-Bildungsstandards-Mathe-Primar.pdf (besucht am 06. 12. 2017).

König, Esther (1989). *Grundkurs PROLOG für Linguisten*. Tübingen: Francke. ISBN: 3-7720-1749-5.

Kumar, E. (2011). *Natural Language Processing*. I.K. International Publishing House. ISBN: 9789380578774.

Landhaeusser, M. (2016). *Eine Architektur fuer Programmsynthese aus natuerlicher Sprache:* Karlsruher Institut für Technologie. ISBN: 9783731505440.

Le, Vu, Sumit Gulwani und Zhendong Su (2013). "SmartSynth". In: *Proceeding of the 11th annual international conference on Mobile systems, applications, and services - MobiSys '13*. ACM Press.

Leiva, Antonio (2016). *Kotlin for Android developers : learn Kotlin the easy way while developing an Android App*. Victoria, British Columbia: Leanpub. ISBN: 978-1530075614.

Lieberman, Henry und Hugo Liu (2006). "Feasibility Studies for Programming in Natural Language". In: *Human-Computer Interaction Series*. Springer Netherlands, S. 459–473.

Liskov, Barbara (1987). "Keynote Address - Data Abstraction and Hierarchy". In: *SIGPLAN Not.* S. 17–34. ISSN: 0362-1340.

Louis, D. und P. Müller (2016). *Android: Der schnelle und einfache Einstieg in die Programmierung und Entwicklungsumgebung*. Carl Hanser Verlag GmbH & Company KG. ISBN: 9783446451124.

Maglie, Andrea (2016). *Reactive Java programming*. United States: Apress. ISBN: 978-1-4842-1428-2.

Martin, Robert C. (2003). *Agile Software Development, Principles, Patterns and Practices*. Pearson Education, Limited.

— (2012). *The Clean Architecture*. URL: https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html (besucht am 14. 11. 2017).

— (2018). *Clean architecture : a craftsman's guide to software structure and design*. Boston: Prentice Hall. ISBN: 978-0-13-449416-6.

Meuser, Michael. und Ulrike Nagel (2013). "ExpertInneninterviews - vielfach erprobt, wenig bedacht". In: *Das Experteninterview: Theorie, Methode, Anwendung*. VS Verlag für Sozialwissenschaften, S. 71–94. ISBN: 9783322932709.

Mew, K. (2016). *Android Design Patterns and Best Practice*. Packt Publishing. ISBN: 9781786465917.

Meyer, B. (1988). *Object-oriented software construction*. Prentice-Hall international series in computer science. Prentice-Hall. ISBN: 9780136290490.

Modrow, Eckart und Kerstin Strecker (2016). *Didaktik der Informatik*. Berlin: De Gruyter Oldenbourg. ISBN: 978-3-486-71622-1.

Niedersächsisches Kultusministerium (2017). *Kerncurriculum für die Grundschule Schuljahrgänge 1 – 4. Mathematik*. URL: http://db2.nibis.de/1db/cuvo/datei/druckfassung_kc_ma_gs.pdf.

Ostrander, J. (2012). *Android UI Fundamentals: Develop & Design*. Develop and Design. Pearson Education. ISBN: 9780132929028.

Petko, D. und E. Jürgens (2014). *Einführung in die Mediendidaktik: Lehren und Lernen mit digitalen Medien*. Beltz Pädagogik. Beltz GmbH, Julius. ISBN: 9783407256782.

Pohl, Klaus und Chris Rupp (2011). *Basiswissen Requirements Engineering : Aus- und Weiterbildung zum "Certified Professional for Requirements Engineering"; Foundation Level nach IREB-Standard*. Heidelberg: Dpunkt-Verl. ISBN: 978-3-8986-4771-7.

Preußig, J. (2015). *Agiles Projektmanagement: Scrum, Use Cases, Task Boards & Co.* Haufe TaschenGuide. Haufe Lexware. ISBN: 9783648065198.

Price, David et al. (2000). “NaturalJava: A natural language interface for programming in Java”. In: *International Conference on Intelligent User Interfaces, Proceedings IUI*. ACM, S. 207–211.

Reber, A.S. (1996). *Implicit Learning and Tacit Knowledge: An Essay on the Cognitive Unconscious*. Oxford Psychology Series. Oxford University Press. ISBN: 9780195344479.

Sanchez, Pedro Vicente Gomez (2016). *Interfaces for presenters in MVP are a waste of time!* URL: http://blog.karumi.com/interfaces-for-presenters-in-mvp-are-a-waste-of-time/ (besucht am 03. 12. 2017).

Schmidt-Thieme, Barbara und Hans-Georg Weigand (2015). “Medien”. In: *Handbuch der Mathematikdidaktik*. Hrsg. von Regina Bruder et al. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 461–490.

Schubert, Sigrid und Andreas Schwill (2011). *Didaktik der Informatik*. Heidelberg Berlin: Spektrum Akademischer Verl. ISBN: 978-3-8274-2652-9.

Storz, R. (2009). *Fachdidaktik - Seminar Mathematik*. Pro Business. ISBN: 9783868053838.

Styczynski, Zbigniew, Krzysztof Rudion und Andre Naumann (2017). *Einführung in Expertensysteme Grundlagen, Anwendungen und Beispiele aus der elektrischen Energieversorgung*. Berlin, Heidelberg: Springer Vieweg. ISBN: 978-3-662-53172-3.

Wagenknecht, Christian (2016). *Programmierparadigmen : Eine Einführung auf der Grundlage von Racket*. Wiesbaden: Springer Fachmedien Wiesbaden. ISBN: 978-3-658-14133-2.

Wagner, S. (2013). *Software Product Quality Control*. SpringerLink : Bücher. Springer Berlin Heidelberg. ISBN: 9783642385711.

Wild, E. und J. Möller (2014). *Pädagogische Psychologie*. Springer-Lehrbuch. Springer Berlin Heidelberg. ISBN: 9783642412912.

Wirdemann, Ralf und Johannes Mainusch (2017). *Scrum mit User Stories*. München: Hanser. ISBN: 978-3-446-45077-6.

Anhang

A. Material

A.1. Interview-Leitfaden

Experten-Interview mit einer Lehrkraft an der Grundschule, die den Unterricht verfolgt hat.

Wo sehen Sie im Fach Mathematik Schwierigkeiten im sprachlichen Bereich?

Welche Kompetenzen sehen Sie durch den Einsatz der App gefördert?

Bietet sich der Bereich Geometrie, genauer „Körper“ an, um Kompetenzen wie *Argumentieren* und den Bereich *kausale Zusammenhänge verstehen* zu fördern?

Welche Einschätzung können Sie zum Lernzuwachs geben?

Was hat für die SuS beim Arbeiten/mit dem Umgang mit der App gut funktioniert?

Wo sahen sie Schwierigkeiten bei der Verwendung der App?

Was würden Sie aus didaktischer Sicht, hinsichtlich des eingesetzten Werkzeugs, an der Unterrichtsstunde bemängeln/positiv hervorheben?

Wie stehen Sie zum Einsatz der Smartphones und Tablets im täglichen Unterricht? Können Sie Vor- und Nachteile nennen?

A.2. Fragebogen Schülerinnen und Schüler

Fragebogen für die Wissensbasis **Experten**

1. Welcher Körper hat mehr Flächen? Pyramide oder Quader?

_____

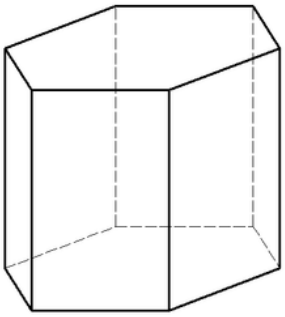
2. Ist jeder Würfel ein Quader? Begründe mit den Eigenschaften.

_____

_____

_____

Dies ist ein Prisma.



3. Nenne mehrere Eigenschaften.

_____

_____

_____

_____

Formuliere eine Regel

_____

5. Nenne zwei Unterschiede zwischen einem Prisma und einer Pyramide.

_____

_____

Male den zutreffenden Smiley aus.

Ich bin gut mit der App zurecht gekommen. 

Ich wusste immer, was ich drücken musste. 

Ich fand das Lernen mit der App interessant. 

Es fiel mir leicht, einzelne Fakten zu erstellen. 

Es fiel mir leicht, einzelne Regeln zu erstellen. 

Ich würde mir wünschen, von so einer App im Unterricht unterstützt zu werden. 

Was hast du beim Arbeiten mit der App Neues gelernt?

_____

_____

Was hat dich am meisten überrascht?

_____

_____

Was hat dich am meisten beim Arbeiten mit der App gestört?

_____

_____

Das möchte ich noch sagen:

_____

_____

Stakeholder	Zielgruppen	Bedürfnisse	Produkt	Wert
• Lernende • Lehrpersonen • Entwickler • Eltern • Schulen • Kultusministerium • Gesetzgeber (Datenschutz)	• Lernende • Lehrpersonen • Entwickler	• Programmieren ohne hohe Einstiegshürden • Moderne Interaktion mit Technik • Einfache Bedienung • Intuitive Bedienung • Wenig Sonderfunktionen • Benutzbarkeit ohne Vorwissen • In den Unterricht integrierbar • Motivierend • Softwarequalität	• Spracheingabe • Smartphone/ Tablet-App • Visuelles Feedback • Sprachliches Feedback • Dokumentation • Architektur	• Spaß am Programmieren • Steigerung Medienkompetenz • Fachliches Verständnis fördern • Fachliche Kompetenzen fördern • Überfachliche Kompetenzen fördern • Geringe Änderungskosten • Schnelle Anpassbarkeit

A.4. Epics und User Stories

Id	Epic	UserStory Beschreibung
8	Mini-Welten	Als Lernender möchte ich eine Übersicht über alle Mini-Welten haben, um eine gespeicherte auswählen zu können
18	Mini-Welten	Als Lernender möchte ich eine Mini-Welt für Fakten und Regeln erstellen, um für jeden Anwendungsfall eine Umgebung zu haben
20	Mini-Welten	Als Lernender möchte ich Mini-Welten editieren, um Eigenschaften zu ändern
21	Mini-Welten	Als Lernender möchte ich Mini-Welten löschen, um nicht mehr benötigte zu entfernen
22	Mini-Welten	Als Lernender möchte ich immer eine Standard-Mini-Welt sehen, um diese als Spielwiese nutzen zu können
19	Wissen aufnehmen	Als Lernender möchte ich Änderungen der Mini-Welt beim erneutem Aufrufen wiedersehen, um nicht jedes Mal die gleichen Eingaben wiederholen zu müssen
5	Wissen aufnehmen	Als Lernender möchte ich der Mini-Welt Fakten hinzufügen, um diese zu erweitern
6	Wissen aufnehmen	Als Lernender möchte ich der Mini-Welt Regeln hinzufügen, um diese zu erweitern
12	Wissen aufnehmen	Als Lernender möchte ich Fakten löschen, um diese aus der Wissensbasis zu entfernen
14	Wissen aufnehmen	Als Lernender möchte ich Regeln löschen können, um diese aus der Wissensbasis zu entfernen
47	Wissen aufnehmen	Als Lernender möchte ich mein erstelltes Wissen sehen, um einen Überblick über den derzeitigen Stand des Systems zu haben
16	Wissen erfragen	Als Lernender möchte ich die Wissensbasis nach Fakten fragen, um die Wahr- oder Unwahrheit dieses Faktes prüfen zu können
17	Wissen erfragen	Als Lernender möchte ich der Wissensbasis ein Ziel vorgeben können, um herauszufinden, ob dieses Wissen vorhanden ist und eine Antwort zu erhalten
40	Intention verstehen	Als Lernender möchte ich Regeln in das System sprechen, um natürlichsprachliche Eingaben zu machen

41	Intention verstehen	Als Lernender möchte ich Fragen in das System sprechen, um natürlichsprachliche Eingaben zu machen
38	Intention verstehen	Als Lernender möchte ich Faktenwissen in das System sprechen, um natürlichsprachliche Eingaben zu machen

A.5. Exemplarische, klassische Unterrichtseinheit zu Körpern

Stunde	Thema	Inhaltlicher/didaktischer Schwerpunkt
1	Erkennen und Benennen geometrischer Körper	Die SuS setzen sich mit den Körpern Würfel, Quader, Kugel, Zylinder, Pyramide und Kegel auseinander und finden sie in der Umwelt wieder.
2	„Was unterscheidet den Würfel vom Quader?“ - Eigenschaften geometrischer Körper	Die SuS erarbeiten die Eigenschaften geometrischer Körper, indem sie Kantenmodelle verschiedener Körper erstellen, diese vergleichen und über ihre Erkenntnisse sprechen.
3	Würfelnetze entdecken und herstellen	Die SuS erweitern ihr räumliches Vorstellungsvermögen, indem sie konkret handelnd sowie in der Vorstellung verschiedene Würfelnetze finden und in eine andere Darstellungsform übertragen.
4	Kopfgeometrische Übungen am Würfel	Die SuS schulen ihr räumliches Vorstellungsvermögen durch gedankliches Operieren mit Würfeln und Würfelnetzen.
5	Baupläne und Würfelgebäude	Die SuS wechseln zwischen der zweidimensionalen und dreidimensionalen Darstellung von Würfelgebäuden, indem sie mit vorgegebenen Bauplänen und Würfelgebäuden arbeiten sowie eigene erstellen.
6	Ansichten	Die SuS vertiefen ihre Raumvorstellungen, indem sie aus verschiedenen Körpern zusammengesetzte Gebäude aus einer bestimmten Ansicht betrachten und in der Vorstellung unterschiedliche Perspektiven einnehmen.
7-8	Lerntheke	Die SuS wenden das zuvor Gelernte selbstständig auf verschiedene Aufgaben an und erweitern ihre Erfahrungen rund um Körper.

A.6. Unterrichtsstunde Fakten Regeln Wissensbasis

Thema	**Lehr/-Sozialform**	**Medien/Orga**	**Zeit**
Thema und Person kennen lernen. Vorstellen, Zielformulierung und Motivation.	Frontalerzählung	Tablet	2’
Fakten und Regeln kennen lernen an Alltagsbeispielen. An der Tafel sammeln was für Regeln es gibt. *Wenn..., dann!*-Formulierung. Beispiel: (Fakt) Max kommt zu spät; (Regel) Wenn Max zu spät kommt, dann muss er sich entschuldigen.	Plenum	Tafel, bunte Kreide	8’
Gruppen werden durch die Lehrperson eingeteilt	Plenum		1’
Verständnis vertiefen was Fakten und Regeln sind. Die SuS spielen gegeneinander Memory. Hierbei achten beide auf eine richtige Zuordnung.	2er-Gruppen	Memorykarten	7’
Nach 6 Minuten: zählen wer gewonnen hat. Gruppen haben Zeit alle Fakten und Regeln zu sortieren. Gruppen, die fertig sind zeigen dies durch Doppelmeldung.	2er-Gruppen		3’

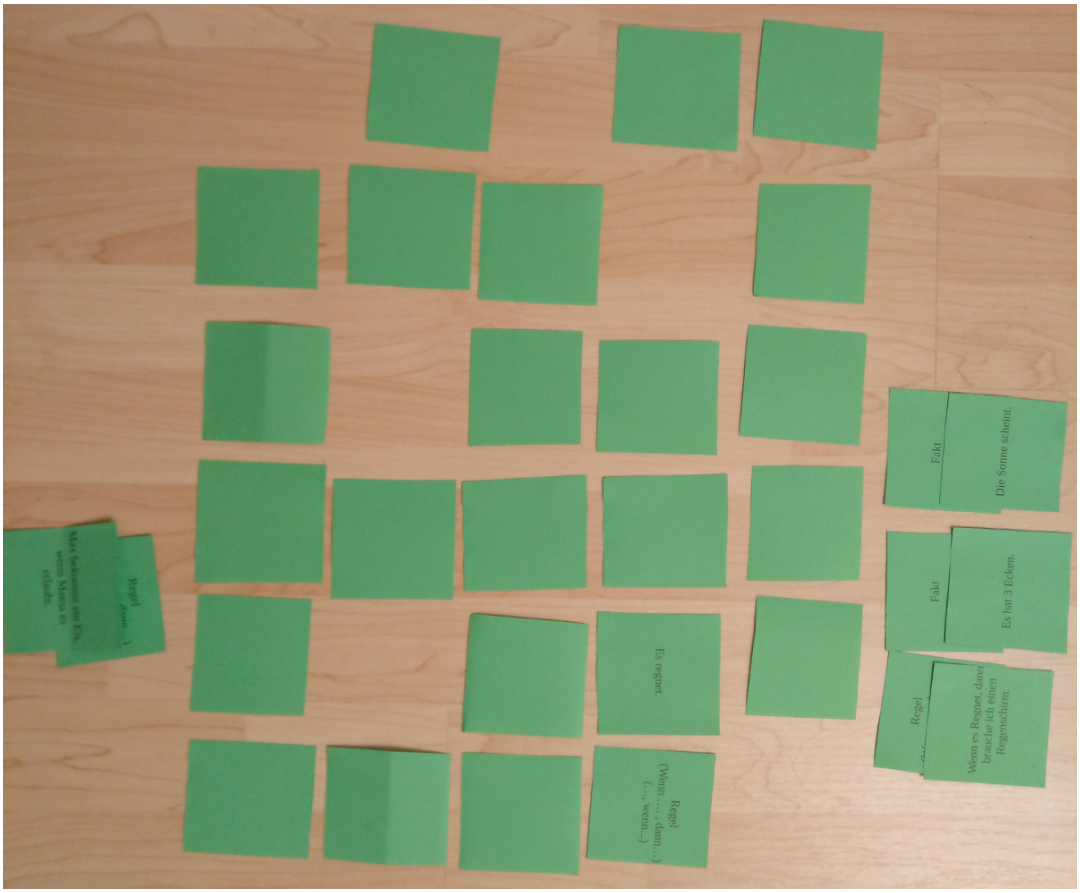
Wissensbasis kennen lernen an Alltagsbeispielen. Es regnet. Es schneit. Die Sonne scheint. Wenn es regnet, brauche ich einen Schirm. Wenn es schneit, brauche ich eine Mütze. Wenn die Sonne scheint, darf ich ein Eis essen. Darf ich ein Eis essen? Brauche ich eine Mütze? Brauche ich einen Schirm? Die Lehrperson erstellt an der Tafel eine Wissensbasis und hängt an ein aufgemaltes Seil, verschiedene Fakten und Regeln und 1) bespricht bzw. 2) erfragt die Auswirkungen	Plenum	Tafel, Vorbereite Moderationskärtchen	8’
Verständnis vertiefen was eine Wissensbasis ist. Die SuS bekommen pro Gruppe eine (sich von den anderen unterscheidende) Frage vorgegeben, die ihre Wissensbasis mit *JA* beantworten soll. Die SuS legen hierfür die Karten aus dem (sortieren) Memory auf dem Tisch an ein Seil.	4er-Gruppen	(sortierte) Memorykarten	8’
Prüfen der Ergebnisse: Sobald eine Gruppe fertig ist, geht sie zum *Meeting-Tisch*. Dort warten sie, bis eine zweite Gruppe fertig ist. Die Gruppen bilden zwei neue 4er-Gruppen mit jeweils zwei Experten für jede Wissensbasis. Sie treffen sich erst bei der einen und dann bei der anderen Wissensbasis und stellen sich diese gegenseitig mit ihrer Frage und Lösung vor. Wichtig ist das Verbalisieren und Argumentieren, warum ihre Wissensbasis die gegebene Frage mit *JA* beantwortet. Anschließend können sich die Gruppen eine weitere Frage holen	4er-Gruppen (jeweils 2 Experten)		8’

A.7. Unterrichtsstunde Körper und Expertensysteme

Thema	**Lehr/-Sozialform**	**Medien/Orga**	**Zeit**
Motivation und Zielformulieren. Der Lehrperson zeigt eine Zielvorstellung anhand einer Mini-Welt.	Stuhl(halb-)kreis (Kinositz)	Tablet	2’
Plakat zur Pyramide vorstellen. Vorwissen reaktivieren durch benennen lassen der Fachbegriffe (*Ecke*, *Kante* und *Fläche*)	Stuhl(halb-)kreis (Kinositz)	Plakat mit Pyramide	2’
Gemeinsames wiederholen: Was sind Fakten? Was sind Regeln? Am Beispiel der Eigenschaften einer Pyramide erläutern. Mini-Welt in der App gemeinsam mit Fakten und Regeln befüllen. SuS dürfen rein sprechen. SuS dürfen die Wissensbasis anfragen	Stuhl(halb-)kreis (Kinositz)	Tablet	5’
SuS bilden 2er/3er Gruppen und probieren die Mini-Welten zu *Mathe-Formen* und *Eis essen* aus, indem die Wissensbasis angefragt und verändert werden kann	Gruppenarbeit	Tablet	5’

Die SuS erhalten ein Hinweisblatt zur Bedienung der App und einen Laufzettel für verschiedene Körper. Ziel ist es in **einer** Mini-Welt, alle Regeln für die Körper (Pyramide, Kugel, Kegel, Würfel, Quader, Zylinder) zu programmieren und dann auf dem Laufzettel abhaken zu lassen. Zu jedem Körper kann sich die Gruppe eine Holz-Figur und das entsprechende Arbeitsblatt nehmen. Die Figur hilft die Fakten und Eigenschaften zu identifizieren. Zum Abhaken des Laufzettels muss die Lehrperson die Regel bestätigen	Gruppenarbeit (Lerntheke)	Körper als Holz-Figuren, Tablets/Smartphones, Laufzettel, Arbeitsblätter, Hilfeblatt	20’
Ein Fragebogen soll den Lernstand der SuS sowie deren Umgang mit der App reflektieren	Einzelarbeit	Fragebogen	10’

A.8. Fakten Regeln Memory



A.9. Fakten Regeln Memory Material

Mama erlaubt ein Eis.	Max ist glücklich, wenn er ein Eis bekommt.	Max bekommt ein Eis, wenn Mama es erlaubt.
Fakt	Regel (Wenn …. , dann…) (…, wenn...)	Regel (Wenn …. , dann…) (…, wenn...)
Ist Max glücklich?	Frage	Ist schönes Wetter?

Ist Peter glücklich?	Die Sonne scheint.	Es regnet.
Frage	Fakt	Fakt
Es schneit.	Fakt	Frage

Wenn es Regnet, dann brauche ich einen Regenschirm.	Wenn die Sonne scheint, dann ist schönes Wetter.	Wird es nass?
Regel (Wenn …. , dann…) (…, wenn...)	Regel (Wenn …. , dann…) (…, wenn...)	Frage
Regent es?	Frage	Es ist ein Quadrat, wenn es 4 Ecken hat und alle Seiten gleich lang sind.

Es ist ein Dreieck, wenn es 3 Ecken hat.	Es hat 4 Ecken.	Es hat 3 Ecken.
Regel (Wenn …. , dann…) (…, wenn...)	Fakt	Fakt
Es ist ein Viereck, wenn es 4 Ecken hat.	Regel (Wenn …. , dann…) (…, wenn...)	Regel (Wenn …. , dann…) (…, wenn...)

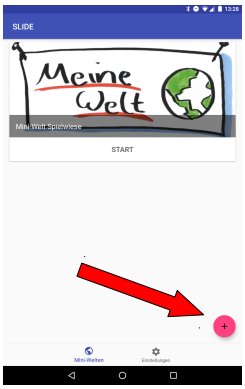
Wenn es keine Ecken hat, dann ist es ein Kreis.	Es hat keine Ecken.	Alle Seiten sind gleich lang.
Regel (Wenn …. , dann…) (…, wenn...)	Fakt	Fakt
Hat es 4 Ecken?	Frage	

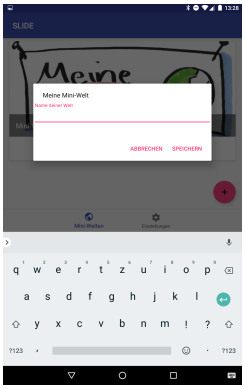
Ist es ein Quadrat?	Ist es Dreieck?	Ist es ein Viereck?
Frage	Frage	Frage

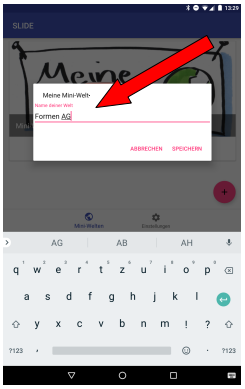
A.10. Arbeitsblatt - Hilfestellung zur App

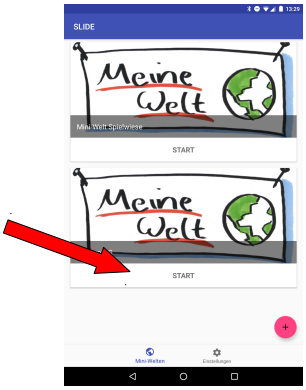
Vorbereitung – Die ersten Schritte!

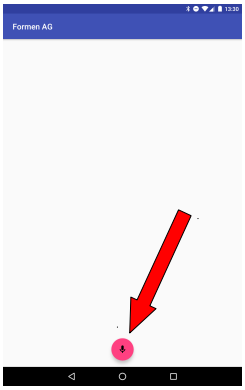
Aufgabe: Erstelle eine Mini-Welt.



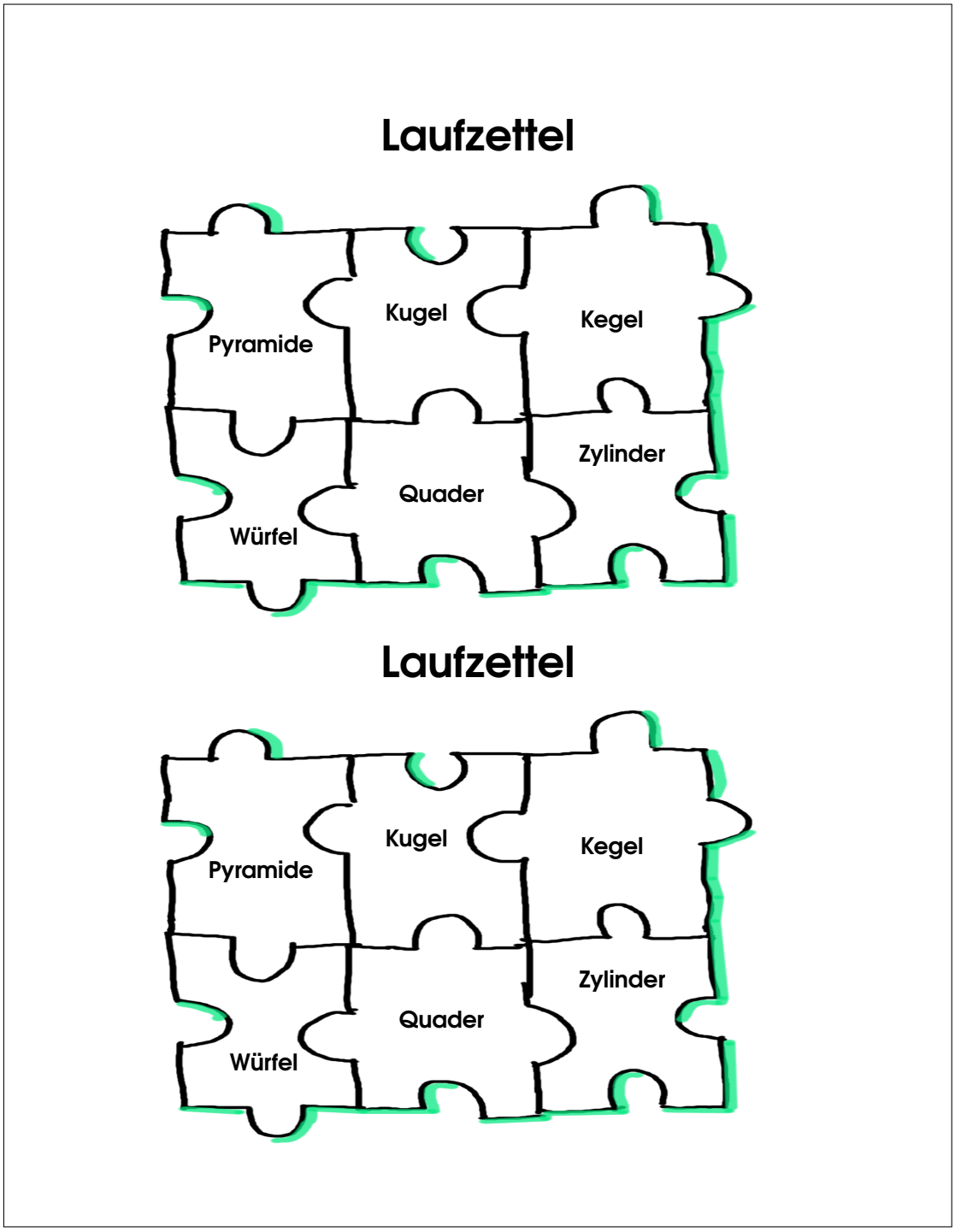








A.11. Arbeitsblatt - Laufzettel



A.12. Aufgabenbeschreibung - Körper

Die Pyramide

Aufgabe 1:
Lösche alle Fakten, die eventuell noch vom vorherigen Körper vorhanden sind. Behalte aber die Regeln.

Aufgabe 2:
Füge Fakten zur Pyramide hinzu. Du darfst den Baustein als Hilfe nutzen.

Aufgabe 3:
Füge eine Regel hinzu die beschreibt, wann die Fakten auf eine Pyramide schließen lassen.

Aufgabe 4:
Frage, ob SLIDE mit den gegebenen Fakten dir bestätigen kann, dass es sich um eine Pyramide handelt. Lass dir vom Lehrer die Form auf dem Laufzettel abhaken.

Die Kugel

Aufgabe 1:
Lösche alle Fakten, die eventuell noch vom vorherigen Körper vorhanden sind. Behalte aber die Regeln.

Aufgabe 2:
Füge Fakten zur Kugel hinzu. Du darfst den Baustein als Hilfe nutzen.

Aufgabe 3:
Füge eine Regel hinzu die beschreibt, wann die Fakten auf eine Kugel schließen lassen.

Aufgabe 4:
Frage, ob SLIDE mit den gegebenen Fakten dir bestätigen kann, dass es sich um eine Kugel handelt. Lass dir vom Lehrer die Form auf dem Laufzettel abhaken.

Der Kegel

Aufgabe 1:
Lösche alle Fakten, die eventuell noch vom vorherigen Körper vorhanden sind. Behalte aber die Regeln.

Aufgabe 2:
Füge Fakten zum Kegel hinzu. Du darfst den Baustein als Hilfe nutzen.

Aufgabe 3:
Füge eine Regel hinzu die beschreibt, wann die Fakten auf einen Kegel schließen lassen.

Aufgabe 4:
Frage, ob SLIDE mit den gegebenen Fakten dir bestätigen kann, dass es sich um einen Kegel handelt. Lass dir vom Lehrer die Form auf dem Laufzettel abhaken.

Der Würfel

Aufgabe 1:
Lösche alle Fakten, die eventuell noch vom vorherigen Körper vorhanden sind. Behalte aber die Regeln.

Aufgabe 2:
Füge Fakten zum Würfel hinzu. Du darfst den Baustein als Hilfe nutzen.

Aufgabe 3:
Füge eine Regel hinzu die beschreibt, wann die Fakten auf einen Würfel schließen lassen. Denke über den Zusammenhang zwischen Quader und Würfel nach.

Aufgabe 4:
Frage, ob SLIDE mit den gegebenen Fakten dir bestätigen kann, dass es sich um einen Würfel handelt. Lass dir vom Lehrer die Form auf dem Laufzettel abhaken.

Der Quader

Aufgabe 1:
Lösche alle Fakten, die eventuell noch vom vorherigen Körper vorhanden sind. Behalte aber die Regeln.

Aufgabe 2:
Füge Fakten zum Quader hinzu. Du darfst den Baustein als Hilfe nutzen.

Aufgabe 3:
Füge eine Regel hinzu die beschreibt, wann die Fakten auf einen Quader schließen lassen. Denke über den Zusammenhang zwischen Quader und Würfel nach.

Aufgabe 4:
Frage, ob SLIDE mit den gegebenen Fakten dir bestätigen kann, dass es sich um einen Quader handelt. Lass dir vom Lehrer die Form auf dem Laufzettel abhaken.

Der Zylinder

Aufgabe 1:
Lösche alle Fakten, die eventuell noch vom vorherigen Körper vorhanden sind. Behalte aber die Regeln.

Aufgabe 2:
Füge Fakten zum Zylinder hinzu. Du darfst den Baustein als Hilfe nutzen.

Aufgabe 3:
Füge eine Regel hinzu die beschreibt, wann die Fakten auf einen Zylinder schließen lassen.

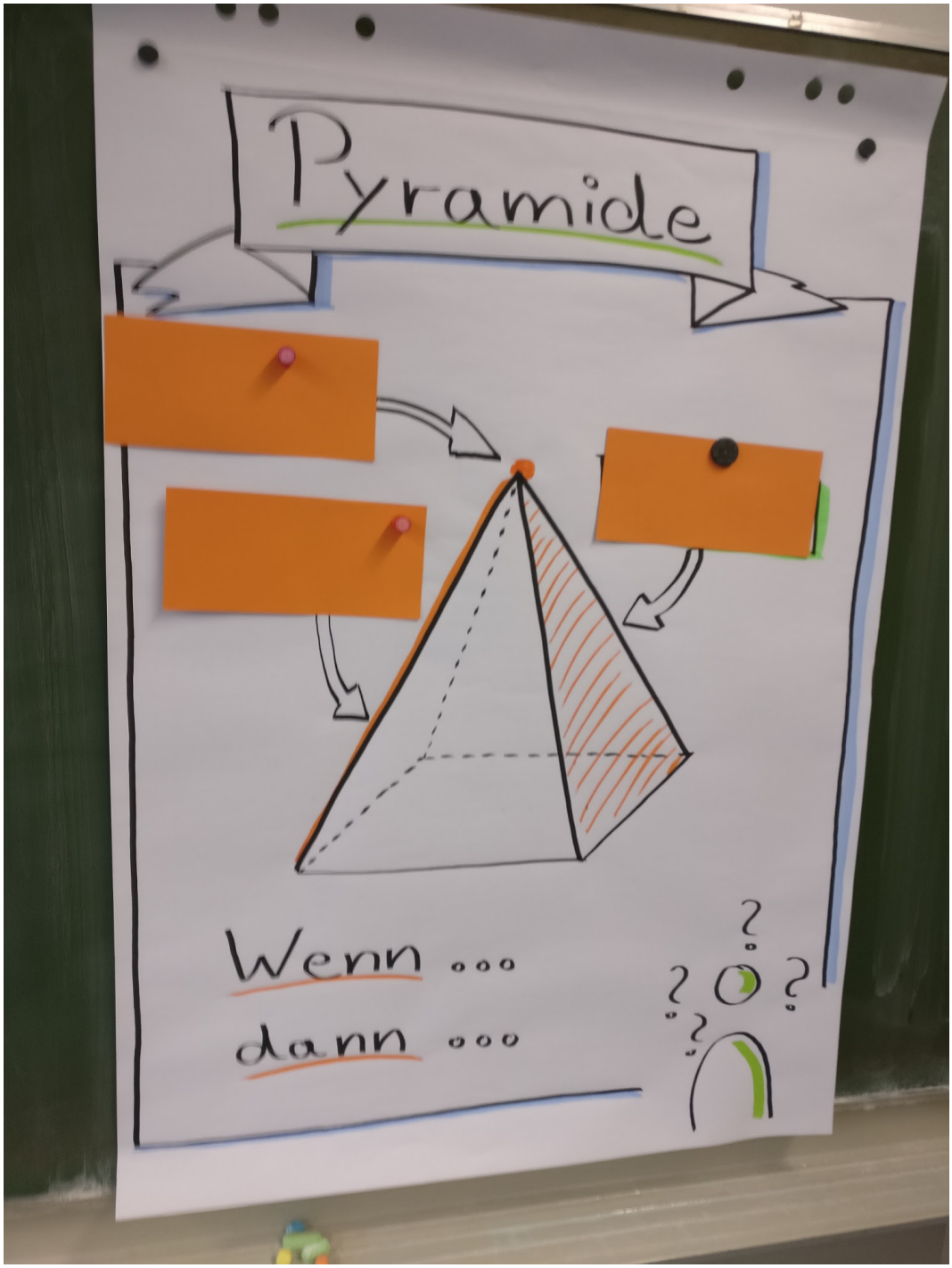
Aufgabe 4:
Frage, ob SLIDE mit den gegebenen Fakten dir bestätigen kann, dass es sich um einen Zylinder handelt. Lass dir vom Lehrer die Form auf dem Laufzettel abhaken.

Abschluss

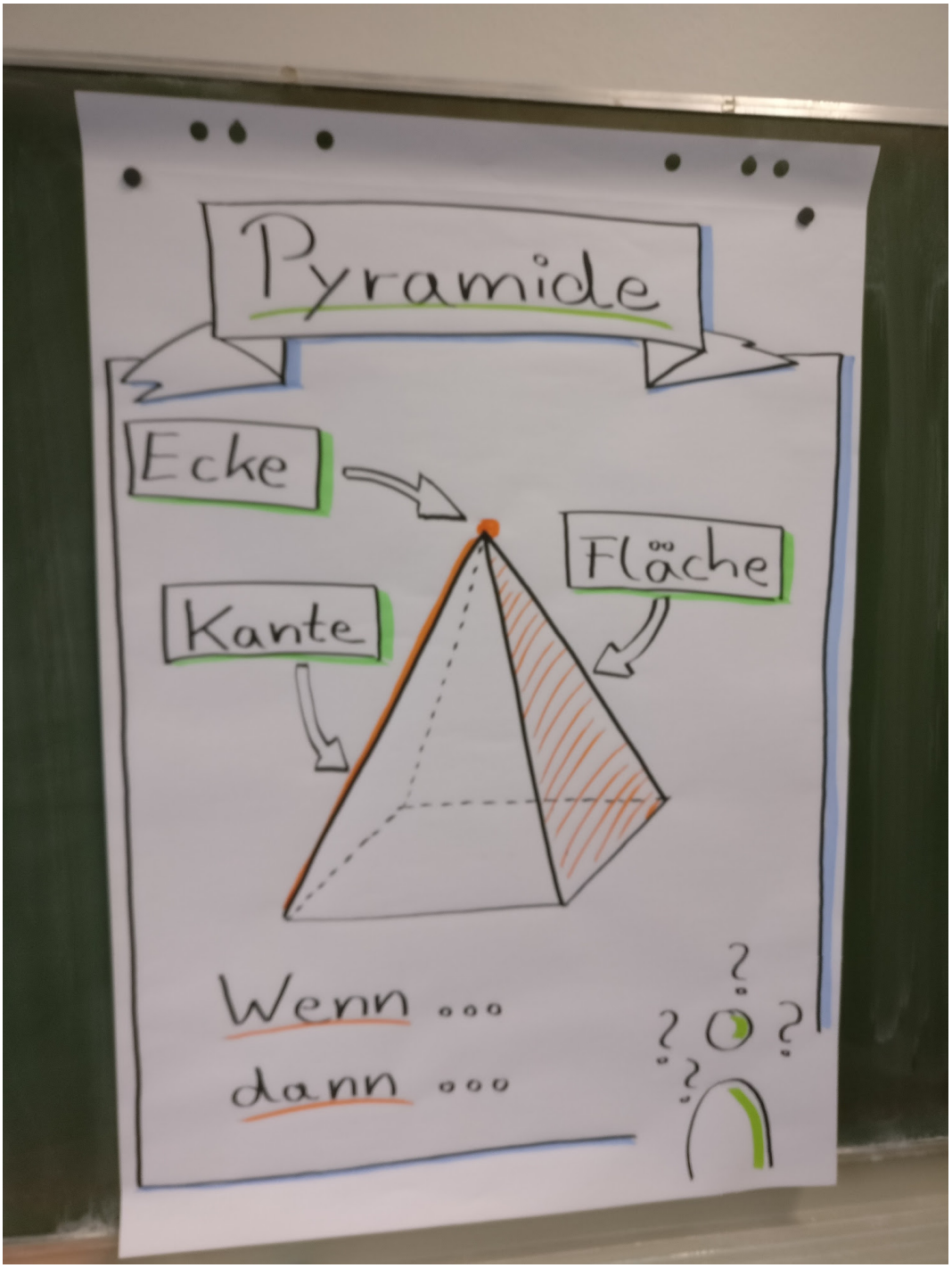
Super! Du hast SLIDE alle Formen beigebracht. Versuche verschiedene Fakten Kombinationen. Du kannst SLIDE fragen, ob es eine bestimmte Form ist.

Aufgabe:
Ist ein Würfel ein Quader? Bespreche und diskutiere mit deinem Partner. Füge die Fakten zum Würfel hinzu. Frage, ob es ein Würfel ist. Frage ob es ein Quader ist.

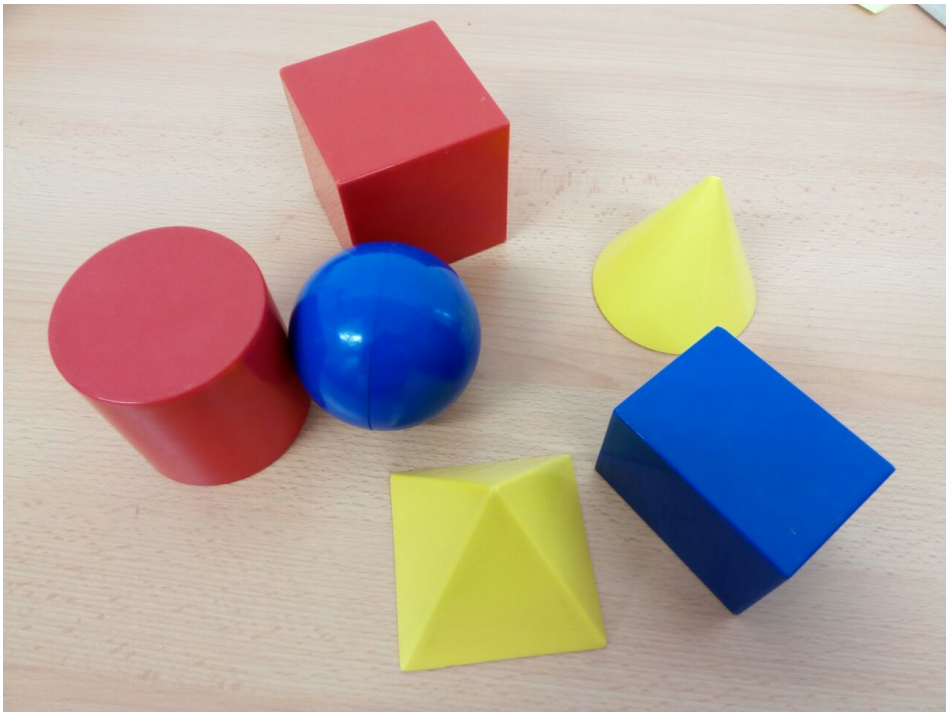
A.13. Plakat zur Pyramide



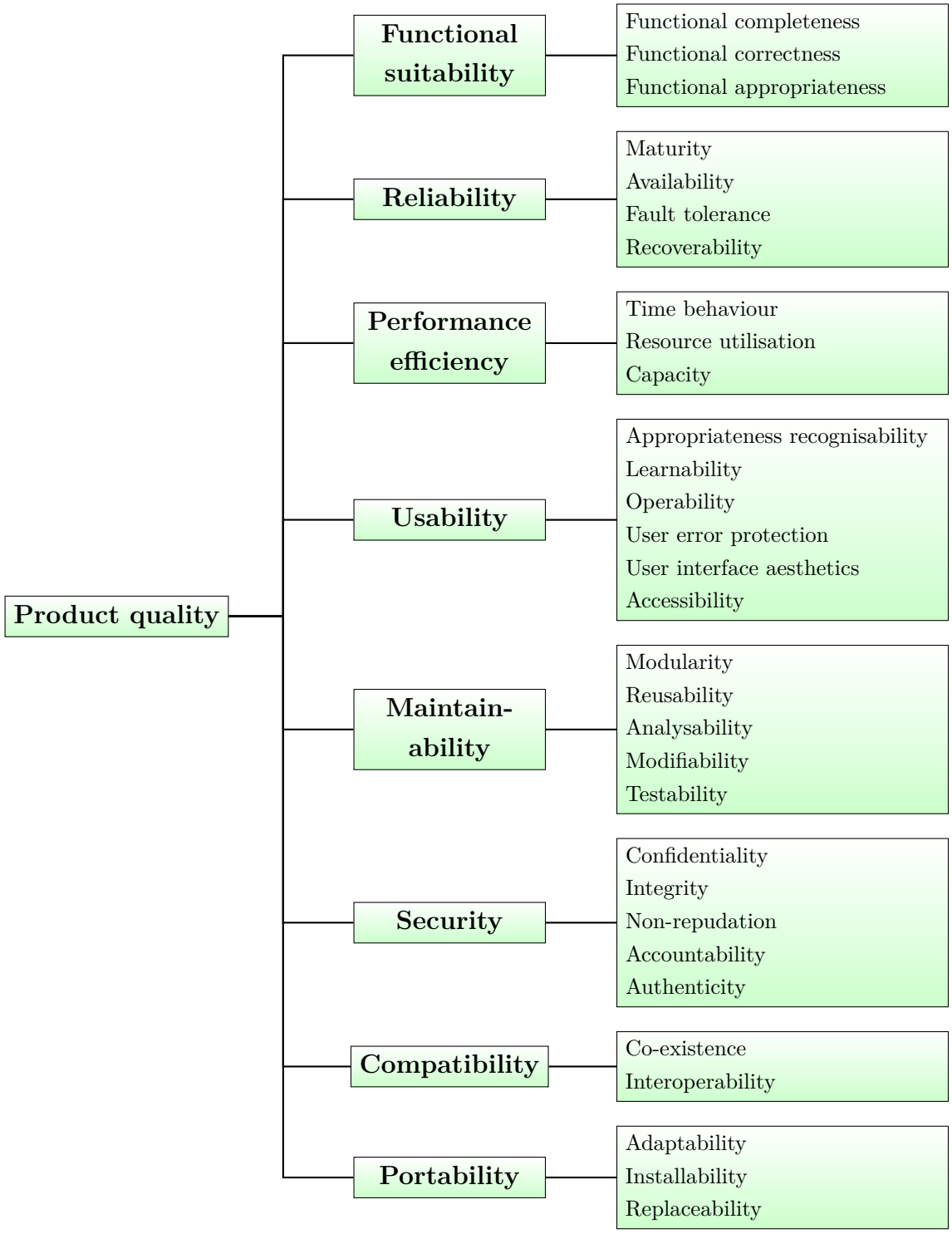
A.14. Plakat zur Pyramide 2



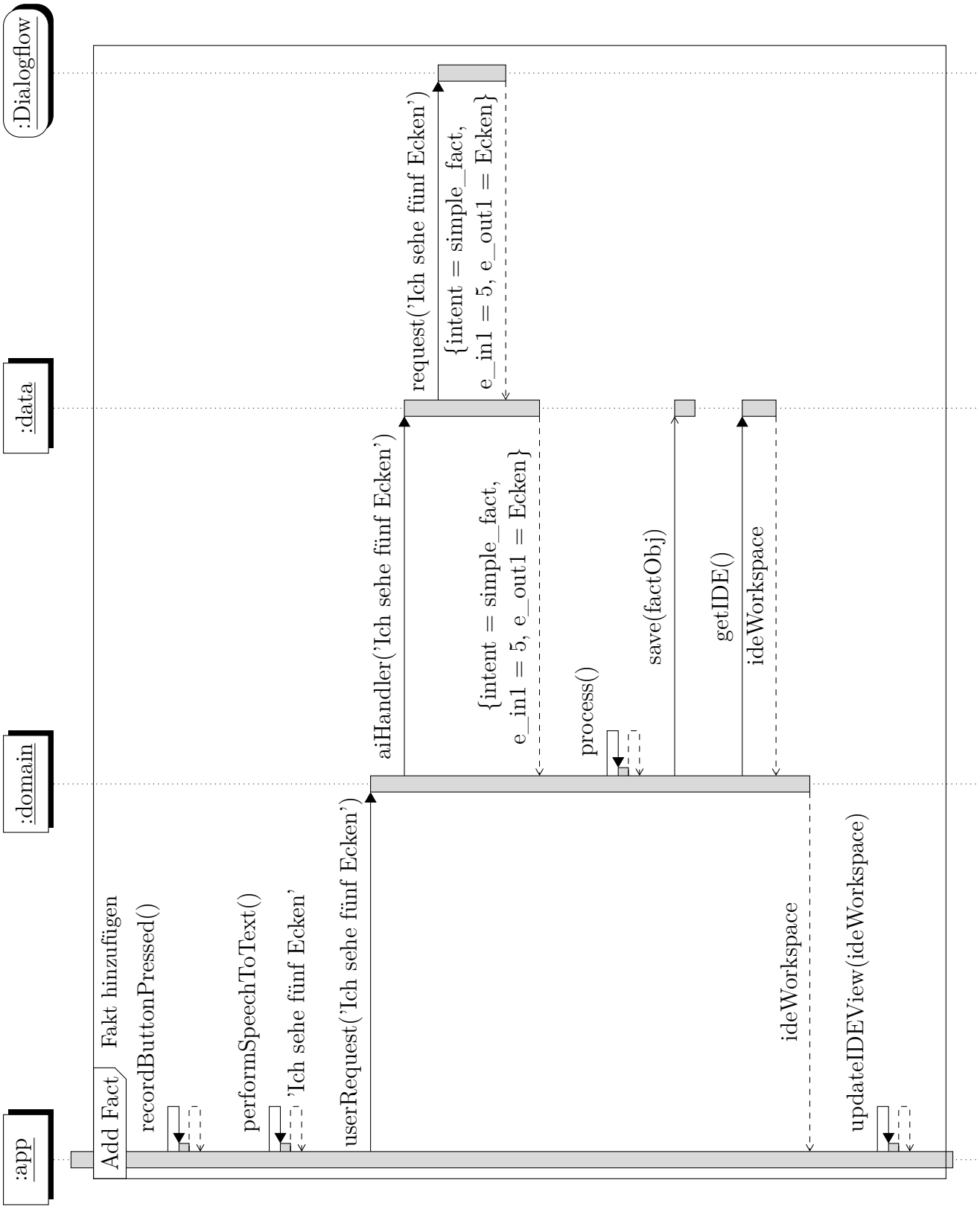
A.15. Hilfsmittel Figuren - Körper



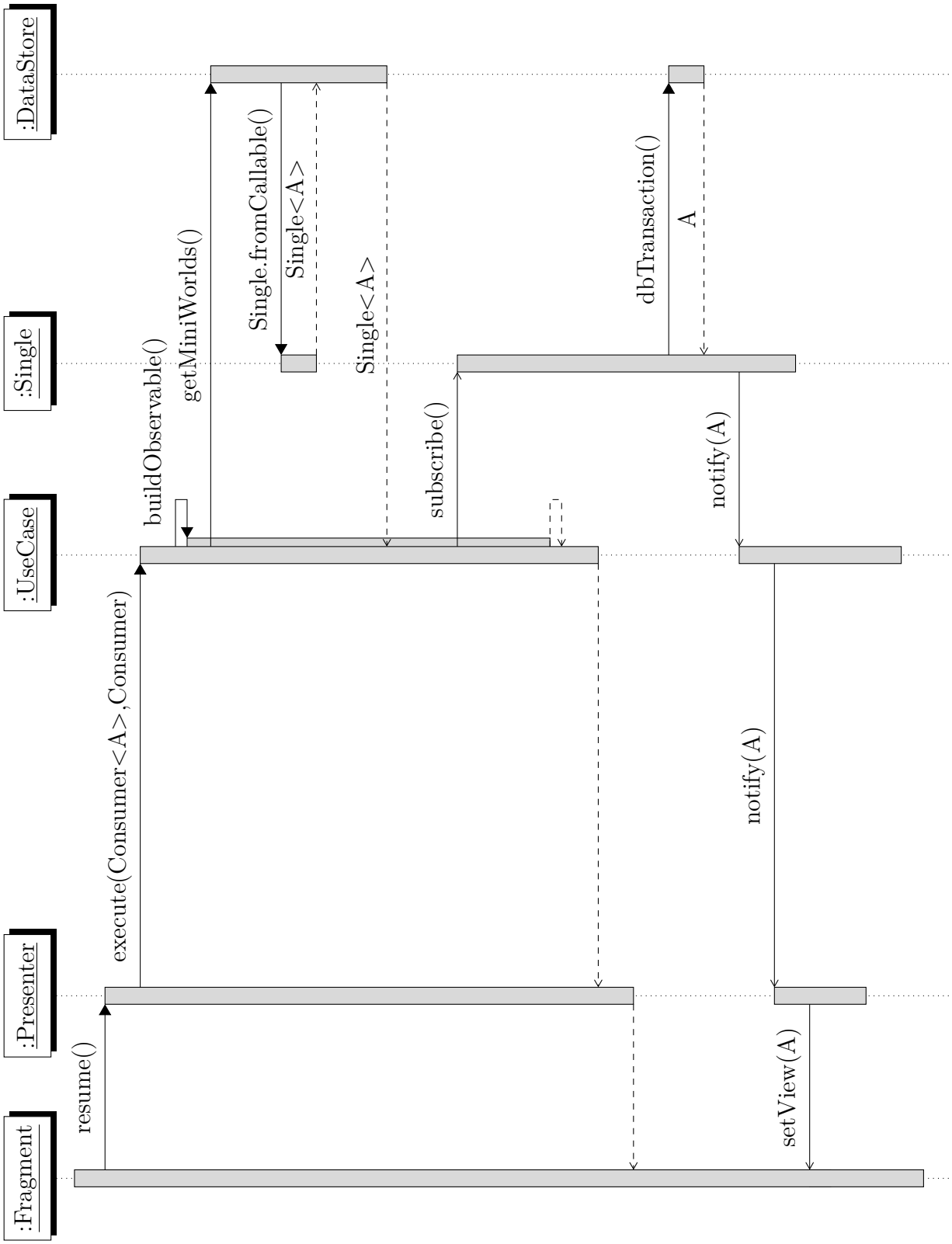
A.16. Product Quality Model nach (ISO/IEC, 2011)



A.17. Fakt hinzufügen SLIDE und Dialogflow



A.18. Kontrollfluss am Beispiel



A.19. Transkriptionsregeln in Modulen nach Fuß und Karbach (vgl. 2014, S. 37ff) Kapitel 4

Sprachglättung	Der Grad, indem die gesprochene Sprache in orthografisch korrekte Schriftsprache übertragen wird. In der sozialwissenschaftlichen Transkription sind drei Stufen etabliert.	
	Keine Glättung	Original gesprochene Sprache
	Leichte Glättung	Annäherung an die Standardorthografie (Beispiel Korrektur eines Dialekts)
	Vollständige Glättung	Beseitigen von Fehlern in Satzbau, Ausdrücken - Beibehaltung von umgangssprachlicher Ausdrucksweise
Pause	Pausen im Gesprächsverlauf können unter anderem in intervallskalierter Form angegeben werden.	
	--	Bis zu zwei Sekunden
	---	Bis zu fünf Sekunden
	(Pause)	Pause ab fünf Sekunden
Sprachklang	Berücksichtigt die Betonung.	
Betonung	immer unbedingt	Betontes Wort Betonte Silbe
Dehnung	ja::: nie:::mals	Gedehntes Wort Gedehntes Silbe
Lautstärke	**niemals** *niemals*	Lauter gesprochen Leiser gesprochen
Nicht-sprachliche Ereignisse	Berücksichtigt die non-verbalen Ereignisse.	
	(räuspert sich) (seufzt) (lacht)	Non-verbale Äußerungen. mit (+) wird das Ende der dazugehörige verbalen Äußerung angegeben
	(haut auf den Tisch)	Hörbare Handlung
	(Telefon klingelt)	Hintergrundgeräusche
Unsicherheit, Unterbrechung, Auslassung	Unterbrechung in der Transkription und unverständliche Passagen	

	(..?) (...??)	Unverständliches Wort oder unverständliche Worte
	(mein?) (mein?/dein?)	Vermutetes Wort
	[...]	Nicht transkribierte Gesprächssequenz
	[...] #00:03:36 bis 00:08:58#	Nicht transkribierte Gesprächssequenz mit Zeitangabe
Zeichensetzung	Orientierung an der deutschen Rechtschreibung	
	.	Satzende
	,	Aufzählung oder Nebensätze
	?	Frage
	:	Wörtliche Rede
	...	Unvollendete Abschnitte
Interaktion	Hörbare Interaktion zum gleichzeitigen Sprechen	
	I Ist das ⌊immer so? B ⌊Ja, das ist es.	Gleichzeitiges Sprechen in der Partiturschreibweise ⌊

B. Quellcode

B.1. GetIDEWorkspaceUseCaseTest.kt

```
1  class GetIDEWorkspaceUseCaseTest {
2   private lateinit var getIDEWorkspaceUseCase:
     ↪ GetIDEWorkspaceUseCase
3   private lateinit var ideRepository: IDERepository
4   private lateinit var testObserver:
     ↪ TestObserver<IDEWorkspaceModel>
5   private lateinit var threadExecutor: ThreadExecutor
6   private lateinit var postExecutionThread:
     ↪ PostExecutionThread
7   @Before
8   fun setUp() {
9    threadExecutor = Mockito.mock(ThreadExecutor::class.java)
10   postExecutionThread =
      ↪ Mockito.mock(PostExecutionThread::class.java)
11   ideRepository = Mockito.mock(IDERepository::class.java)
12   getIDEWorkspaceUseCase =
      ↪ GetIDEWorkspaceUseCase(threadExecutor,
      ↪ postExecutionThread, ideRepository)
13   testObserver = TestObserver()
14  }
15  @Test
16  fun shouldGetVolatileWorkspace() {
17   Mockito.`when`(ideRepository⌋
      ↪ .getVolatileIDEWorkspace(DomainTestData⌋
      ↪ .miniWorldProfileVolatileTestModel,false))⌋
      ↪ .thenReturn(Observable.just(DomainTestData⌋
      ↪ .iDEWorkspaceVolatileTestModel))
18   getIDEWorkspaceUseCase⌋
      ↪ .buildUseCaseObservable(GetIDEWorkspaceUseCase.Params⌋
      ↪ .forWorkspace(DomainTestData⌋
      ↪ .miniWorldProfileVolatileTestModel))⌋
      ↪ .subscribe(testObserver)
```

```
19
20   Mockito.verify(ideRepository,Mockito.times(1))⌋
       ↪  .getVolatileIDEWorkspace(DomainTestData⌋
       ↪  .miniWorldProfileVolatileTestModel,
       ↪  false)
21   Mockito.verifyNoMoreInteractions(ideRepository)
22   testObserver.assertValue(DomainTestData⌋
       ↪  .iDEWorkspaceVolatileTestModel)
23   testObserver.assertComplete()
24  }
25  @Test
26  fun shouldGetPersistenceWorkspace() {
27   Mockito.`when`(ideRepository⌋
       ↪  .getIDEWorkspace(DomainTestData⌋
       ↪  .miniWorldProfileTestModel,false))⌋
       ↪  .thenReturn(Observable.just(DomainTestData⌋
       ↪  .iDEWorkspaceTestModel))
28   getIDEWorkspaceUseCase⌋
       ↪  .buildUseCaseObservable(GetIDEWorkspaceUseCase.Params⌋
       ↪  .forWorkspace(DomainTestData⌋
       ↪  .miniWorldProfileTestModel)).subscribe(testObserver)
29   Mockito.verify(ideRepository,Mockito.times(1))⌋
       ↪  .getIDEWorkspace(DomainTestData⌋
       ↪  .miniWorldProfileTestModel,
       ↪  false)
30   Mockito.verifyNoMoreInteractions(ideRepository)
31   testObserver.assertValue(DomainTestData⌋
       ↪  .iDEWorkspaceTestModel)
32   testObserver.assertComplete()
33  }
34 }
```

B.2. GetIDEWorkspaceUseCase.kt

```
class GetIDEWorkspaceUseCase @Inject
 ↪ constructor(threadExecutor: ThreadExecutor,
 ↪ postExecutionThread: PostExecutionThread, private val
 ↪ ideRepository: IDERepository) :
 ↪ ObservableUseCaseWithParameter<IDEWorkspaceModel?,
 ↪ GetIDEWorkspaceUseCase.Params>(threadExecutor,
 ↪ postExecutionThread) {

 override fun buildUseCaseObservable(params: Params):
  ↪ Observable<IDEWorkspaceModel?> {
  return if(params.miniWorldProfileModel?.profileId == 1){
    this.ideRepository.getVolatileIDEWorkspace(params
     ↪ .miniWorldProfileModel,
     ↪ false)
   }else{
        this.ideRepository.getIDEWorkspace(params
         ↪ .miniWorldProfileModel,
         ↪ false)
   }
 }

 class Params private constructor(val miniWorldProfileModel:
  ↪ MiniWorldProfileModel?) {
  companion object{
   fun forWorkspace(miniWorldProfileModel:
    ↪ MiniWorldProfileModel?): Params {
    return Params(miniWorldProfileModel)
   }
  }
 }
}
```

B.3. MiniWorldProfileViewModel.kt

```
data class MiniWorldProfileViewModel(
 val profileId: Int,
 val profileType: ProfileType,
 var description: String?,
 val image: Int,
 var isSelected: Boolean = false) : Parcelable {

 companion object{
  @JvmField val CREATOR:
   ↪ Parcelable.Creator<MiniWorldProfileViewModel> =
   ↪ object:Parcelable.Creator<MiniWorldProfileViewModel> {
   override fun createFromParcel(source: Parcel):
    ↪ MiniWorldProfileViewModel {
    return MiniWorldProfileViewModel(source)
   }
   override fun newArray(size: Int):
    ↪ Array<MiniWorldProfileViewModel?> {
    return arrayOfNulls(size)
   }
  }
 }

 constructor(parcelIn: Parcel) : this(
  parcelIn.readInt(),
  if(parcelIn.readInt() == 0) ProfileType.COMMON else
   ↪ ProfileType.TMP,
  parcelIn.readString(),
  parcelIn.readInt(),
  parcelIn.readInt() != 0
 )

 override fun writeToParcel(dest: Parcel?, flags: Int) {
  dest?.writeInt(profileId)
  dest?.writeInt(profileType.type)
```

```
  dest?.writeString(description)
  dest?.writeInt(image)
  dest?.writeInt(if (isSelected) 1 else 0)
 }

 override fun describeContents(): Int = 0
}
```

B.4. SingleFactView.kt

```
class SingleFactView(
 context: Context, override var item: Knowledge?
 ) : FactView(context, item), KnowledgeView {

 override val params = RelativeLayout.LayoutParams(ViewGroup
  ↪ .LayoutParams.WRAP_CONTENT,
  ↪ ViewGroup.LayoutParams.WRAP_CONTENT)

 constructor(
  context: Context, item: Knowledge, rule: Boolean
 ) : this(context, item as SingleFact){
  init(rule)
 }

 init {init(false)}

 override fun init(rule: Boolean) {
  super.init(rule)
  text = (item as SingleFact).name
 }

 override fun onDraw(canvas: Canvas) {
  super.onDraw(canvas)
 }

 override fun style() {
  super.style()
 }
}
```

B.5. PropertyFactView.kt

```
1  class PropertyFactView(context: Context, override var item:
    ↪ Knowledge?, rule: Boolean) : FactView(context, item) ,
    ↪ KnowledgeView {
2
3   override val params = RelativeLayout.LayoutParams(ViewGroup
    ↪ .LayoutParams.WRAP_CONTENT,
    ↪ ViewGroup.LayoutParams.WRAP_CONTENT)
4
5   constructor(context: Context, fact: Knowledge):
    ↪ this(context, fact, false){
6    init(false)
7   }
8
9   init {
10   init(rule)
11  }
12
13  override fun init(rule: Boolean) {
14   super.init(rule)
15   val fact = item as PropertyFact
16   val builder = StringBuilder()
17   if (fact.facts.isNotEmpty()) {
18    builder.append((fact.facts[0] as SingleFact).name)
19    builder.append(" ")
20   }
21   builder.append((fact.simpleFact as SingleFact).name)
22   fact.facts.forEachIndexed { index, knowledge ->
23    if (index != 0) {
24     builder.append(" ")
25     builder.append((knowledge as SingleFact).name)
26    }
27   }
28   text = builder.toString()
29  }
```

```
30
31  override fun onDraw(canvas: Canvas) {
32   super.onDraw(canvas)
33  }
34
35  override fun style() {
36   super.style()
37  }
38 }
```

B.6. IDEContract.kt

```
interface IDEContract {
 interface View : BaseView {
  var ideWorkspaceViewModel: IDEWorkspaceViewModel?
  fun setIDEViewModel(ideWorkspaceViewModel:
   ↪ IDEWorkspaceViewModel?)
  fun getProfileModel(): MiniWorldProfileViewModel?
  fun getSelectedKnowledge(): List<KnowledgeView>?
  fun setKnowledgeUnSelected()
 }
 interface Presenter : BasePresenter<View> {
  fun performUserInput()
  fun deleteKnowledge()
 }
}
```

B.7. updateMiniWorlds() in MiniWorldsPresenter.kt

```
override fun updateMiniWorlds() {
 getMiniWorldsUseCase.execute(
  Consumer {
   view?.setMiniWorlds(miniWorldsMMapper.transformMtoVM(it))
  },
  Consumer {
   if (it is EmptyDatabase) {
    createDefaultMiniWorldProfile()
   } else {
    view?.error(it.message!!)
   }
  }
 )
}
```

B.8. execute() in SingleUseCase.kt

```
internal abstract fun buildUseCaseObservable(): Single<T>

fun execute(success: Consumer<T>, error: Consumer<Throwable>)
 ↪ {
 Preconditions.checkNotNull(success)
 Preconditions.checkNotNull(error)
 val observable = this.buildUseCaseObservable()
  .subscribeOn(Schedulers.from(threadExecutor))
  .observeOn(postExecutionThread.scheduler)
 addDisposable(observable.subscribe(success, error))
}
```

B.9. buildUseCaseObservable() in GetMiniWorldsUseCase.kt

```
override fun buildUseCaseObservable():
 ↪ Single<MiniWorldsModel> {
 return this.repository.getMiniWorlds()
}
```

B.10. getMiniWorldsData() und getMiniWorlds() in MiniWorldsDataStore.kt

```
override fun getMiniWorlds(): Single<MiniWorldsModel> {
 logger.info { "DATASTORE -> getMiniWorlds" }
 return getMiniWorldsData().map {
  ↪ miniWorldMMapper.transformDMtoM(it) }
}

private fun getMiniWorldsData(): Single<MiniWorldsDataModel>
 ↪ {
 return Single.fromCallable({
 if ((database.find(MiniWorldProfileDataModel()) as
  ↪ MutableList<*>).isEmpty()) {
  throw EmptyDatabase("Database is Empty")
 }
 val list = database.find(MiniWorldProfileDataModel())
 val profileList = list.map { it as MiniWorldProfileDataModel
  ↪ }
 MiniWorldsDataModel(profileList)
 })
}
```

B.11. ApiAiServiceHandler.kt

```
class ApiAiServiceHandler @Inject constructor() :
 ↪ APIAIHandler, AIListener {

 @Inject lateinit var logger: AnkoLogger
 @Inject lateinit var aiService: AIService
 private var response: AIResponse? = null

 override fun getApiAiResponse(result: AIResponse?):
  ↪ Observable<AIResponse?> {
  response = result
  if (response == null) {
   aiService.setListener(this)
   aiService.startListening()
  }
  return Observable.defer({ Observable.fromCallable {
   ↪ response } })
   .retry({
   _, error -> error is NullPointerException
  })
 }

 override fun onResult(result: AIResponse?) {
  response = result
 }
...
```

C. Ergebnisse

C.1. Experteninterview Transkript (E1)

Pfad zur Datei (Datenträger)	Aufnahmen/ExpertenInterview1.mp3
Zeitstempel	27.11.2017 14:35
Ort	Poggenhagen
Dauer	21 Minuten, 48 Sekunden
Datenerhebung	Leitfadeninterview - Experteninterview
Befragte/r	Sonja Teichmann (T)
Interviewer/in	Marcel Kaufmann (K)
Transkriptionsregeln	Die Transkription erfolgte Wort für Wort ohne Sprachglättung und ohne Füllwörter (äh, ähm, ...) Die Regeln sind in Abschnitt 7.2 detailliert dargestellt.
Verwendete Transkriptionsmodule	- Sprachglättung - Sprachklang - Zeichensetzung

K: Ja, Hallo Frau Teichmann.
T: Hallo.
K: Danke erstmal, dass Sie sich bereit erklärt haben, mit mir dieses Interview zu führen. Sie sind die Lehrkraft in der Programmier-AG, richtig?
T: Ja
K: Welche Fächer haben Sie studiert?
T: Mathematik und Sachunterricht.
K: Ok, dann kommen wir auch gleich zur ersten Fragen. Wo sehen sie im Fach Mathematik die Schwierigkeiten im sprachlichen Bereich?
T: Erstmal kann man sagen, dass ja Schwierigkeiten überall auftreten können, alleine schon weil Kinder in ihrem Alltag ganz anders sprechen als vor allem in Fächern. Das ist in Mathe vor allem so, weil ganz viele Fachbegriffe vorhanden sind und ja, diese Differenzen zwischen der Alltagssprache, überhaupt unserer Bildungssprache in der

Schule, und dann dem Fachwortschatz in dem Fach ja, zu großen Problemen kommen können.

K: Und, ich habe jetzt ja eine App entwickelt, die sich ja auf einige Problemstellungen beziehen soll und dort unterstützen soll. Jetzt so von außen betrachtet und Sie waren ja auch mit in der Unterrichtseinheit, welche Kompetenzen würden Sie sagen, sehen Sie durch die App denn gefördert?

T: Also es war auf jeden Fall so, dass die Kinder sehr kooperativ miteinander gearbeitet haben, also dieser ganze soziale Bereich, Sozialkompetenzen, Teamfähigkeit, Kompromissbereitschaft, das stand auf jeden Fall ganz klar im Vordergrund und dann ein weiterer großer Punkt ist natürlich der Umgang mit Medien, der heutzutage eine immer größere Rolle einnimmt und für die Kinder nicht nur, dass man mal in den Computerraum geht und etwas recherchiert, sondern dass man die Medien auch einsetzt, das die Kinder selbstständig damit arbeiten können. Das war hier eben auch ganz klar. Dann der dritte Bereich das wir eben auch schon ein bisschen angesprochen haben sind dann die sprachlichen Kompetenzen, das ist ja vor allem jetzt mit Begriffen und verschiedenen Zusammenhängen aus dem geometrischen Bereich gehandelt wurde und da ja, ging es eben vor allem um Sätze, um Regeln, die man formulieren sollte, das die Kinder auch bewusst gesprochen haben, die einfachen Zusammenhänge dann ja auch durch überlegte Formulierung in die App reinsprechen sollten und wenn es zum Beispiel darum geht die App nochmal nach einem, ja Körper zu fragen oder zu schauen: ist das jetzt eine Kugel, zum Beispiel, oder nicht, dann geht es natürlich auch so ein bisschen um das Argumentieren. Aber der Bereich Kommunizieren, der jetzt sowieso in der Schule immer ganz ganz wichtig ist, ist hier ganz stark vertreten gewesen.

K: OK, und ein anderer Aspekt, der Zusammenhänge, die ich ja auch als Kompetenz angedacht hatte anzusprechen, wo sehen Sie da im Einsatz der App Möglichkeiten? In dem Fall wie es eingesetzt wurde schon? Oder müsste man da jetzt noch etwas verändern, um nochmal stärker kausale Zusammenhänge

zu forcieren?

T: Also hier ging es ja weniger dann darum etwas auch konkret zu belegen, zu begründen, zu hinterfragen warum etwas so ist... Aber die Zusammenhänge zwischen den Körpern, die wurden ja durch die Regeln ganz klar hervorgehoben, das die Kinder wirklich diese wenn, dann Satzbausteine haben, die wir zum Beispiel im Matheunterricht auch ganz häufig wieder finden können wenn man dann eben auch sagt es ist eine Gerade wenn der Strich so und so gezeichnet ist, das kennen die Kinder auch und gerade diese ja kausalen Zusammenhänge waren da auf jeden Fall vertreten. Und in der Geometrie kann man da ja dann auch ganz viel auch visuell und durch verschiedene Handlungen mit den Zusammenhängen erstellen und darstellen.

K: Ok, ist also allgemein dieser Bereich der Geometrie auch ja dafür prädestiniert solche Kompetenzen wie Argumentieren kausale Zusammenhänge zu fördern ja gut geeignet?

T: Ja auf jeden Fall es ist so gut wie in jedem Bereich der Mathematik der in der Grundschule abgedeckt wird sehr stark vorhanden aber gerade in der Geometrie durch die Anschauung die dann auch da ist und die unterschiedlichen Handlungsmöglichkeiten die da auch verstärkt sind kann man eben dieses Gefüge noch stärker von verschiedenen Ebenen betrachten und auf verschiedenen Wegen wieder lernen und automatisieren.

K: Ok, ich habe ja die Unterrichtsstunde so ein bisschen außerhalb des Mathematikunterrichts geplant und jetzt auch durchgeführt. Die Themen der Eigenschaften der Körper die habe die Kinder ja schonmal im Mathematikunterricht gehabt und wurde im Unterricht schon behandelt, trotzdem ist ja nochmal die Frage so im Vergleich zum Mathematikunterricht nochmal andere Methodik eingesetzt dort... haben Sie einen besonderen Lernzuwachs erkennen können direkt?

T: Also ich bin der Meinung, dass da eine ganz starke Rückkopplung auch zu den Kompetenzen besteht die wir vorhin schon angesprochen haben, die durch diese App gefördert werden und eben jetzt auch schon wurden in dieser Stun-

de und der Lernzuwachs besteht vor allem daneben, neben diesen sozialen und sprachlichen Punkten dann auch wieder darin, dass hier jetzt ganz stark diese wenn dann Beziehung vorgehoben wurde und sie die dann auch gleich anwenden und trainieren konnten... also dass das Ziel, das die Kinder mit dieser App selbstständig umgehen, da selbstständig etwas reinsprechen und daraus dann eben auch Beziehungen wieder herstellen können und auch überprüfen können wurde da auf jedenfall abgedeckt und ich denke, dass der Lernzuwachs einfach auch ist, dass die Kinder merken, dass nicht nur dieses eine Thema sondern, dass es jetzt auch exemplarisch betrachtet werden kann, dass man mit einer App als Werkzeug oder generell dem Tablet oder Smartphone jetzt auch andere Zusammenhänge sowohl im Matheunterricht als auch in anderen Fächern dann wieder durch das eigene Sprechen ganz stark üben kann. Und für die Kinder dazu dann auch, vielleicht noch über den Lernzuwachs hinaus gehend, dass sie einfach sehen, dass man durch Sprechen ganz viel bewirken kann und auch Lernprozesse wieder noch stärker fokussieren kann.

K: Ok, ich habe den Kindern ja in den Fragebögen, die ich am Ende ausgeteilt habe auch so eine Einschätzung nach einer Einschätzung gefragt, wie sie die ja den Umgang mit der App empfunden haben... welche Schwierigkeiten sie hatten, was gut funktioniert hat aber für mich ist natürlich jetzt auch interessant, von der Lehrkraft, also von Ihnen, die noch einmal einen anderen Blick darauf hat... sie waren ja auch die ganze Zeit mit dabei sind mit in die Gruppen gegangen, haben also hautnah miterlebt wie damit gearbeitet wurde... ja wie haben Sie das empfunden. Was hat denn mit dem Umgang der App gut funktioniert?

T: Also es war auf jeden Fall sehr sehr motivierend für die Kinder. Es ist ja überall im Unterricht immer zu sehen, wenn mal etwas ist was jetzt nicht normal ist, in Häckchen, dass die Kinder dann gleich sehr motiviert sind, um eben auch, ja damit selbst umzugehen und sich damit auseinander zu setzen ein Problem damit anzugehen. Und in

diesem Fall lag es eben auch klar auf der Hand, dass die Handlungsorientierung zu dieser Motivation beigetragen hat dadurch, dass sie diese Körper vor Augen hatten und daran das auch überprüfen konnten, das dann genutzt haben um in die App reinzusprechen und das eben dann auch gleich noch mit einem Partner der dabei war, wo eben auch wieder das Kommunizieren über etwas und das kooperative Lernen im Vordergrund standen, dass dadurch ganz stark der motivierende Charakter da war. Dann, was ich noch ganz toll fand, ist ja, dass wirklich wenig Vorwissen dafür nötig war, dass die Kinder ja sofort mit dieser App starten konnten nach ein paar Erklärungen. Wichtig war natürlich, dass sie da schon ein bisschen gelernt hatten was jetzt Regeln und Fakten sind aber da kommen wir ja dann bestimmt gleich nochmal dazu und für die Kinder war es auch visuell einfach sehr ansprechend und durch diese Anschaulichkeit, die auch durch die klaren Strukturen dann in der App da waren und diese WOW Effekte, dass wenn man jetzt was spricht wird da gleich etwas angezeigt, das ja das Kinder auch einfach nochmal dazu anregt sich noch weiter damit zu beschäftigen.

K: Und bei all dem positiven war das jetzt ja der erste Entwurf, das erste Ausprobieren mit so einer Art zu arbeiten... Haben Sie direkt Schwierigkeiten gesehen, die die Kinder beim Verwendung der App hatte?

T: Ja also es kam auch von den Kindern schon ein paar, ich sag mal Anregungen, was nicht so gut geklappt hat. Sie haben zum Beispiel selber festgestellt, und das konnte ich auch sehr gut beobachten, dass es wichtig ist, dass man sehr genau spricht, dass die App eben wenn man, ja Wörter verschluckt oder so im Alltag vor sich hin redet, dass dann manchmal nicht so genau erkennen kann. Dann war es ganz wichtig, dass es einfach nicht zu laut ist der Umgebung, da muss man eben darauf achten, das ist natürlich im realen Schulalltag manchmal nicht zu bewerkstelligen wenn man 30 Kinder in der Klasse sitzen hat oder auch 23, dass es dann wirklich ruhig ist und funktioniert das müsste

L

man gut einüben aber ich kann mir das durchaus vorstellen, dass es möglich ist und dann hatte ich auch einen Schüler dabei, der ja, damit jetzt nicht gut umgehen konnte, weil er letztes mal nicht dabei war und Regeln und Fakten jetzt nicht verstanden hatte und für ihn war es schwierig ohne Vorwissen damit umzugehen. Dahingegen gab es einen anderen Schüler, der trotzdem gut damit klar gekommen ist und ich denke, dass ja deswegen ganz unterschiedlich sein aber so ein gewisser Grundstein ist schon sinnvoll. Und ein letzter Punkt, der mir aufgefallen ist, dass diese Sprachsensibilität liegt eben im Fokus und in dieser Lerngruppe war es jetzt so, dass keine Kinder da sind die Sprachschwierigkeiten haben, die aus einem Haushalt kommen mit Migrationshintergrund aber da kann ich mir jetzt auch so vorstellen, dadurch das die Lerngruppen ja immer heterogener werden, dass es da da eben auch schwierig sein kann, wenn man allein schon ganze Sätze sprechen soll. Aber ich kann mir auch vorstellen, dass es eine Möglichkeit ist, dieses zu fördern also dem wieder entgegenzuwirken und gerade das zu nutzen.

K: Jetzt haben wir ja sehr stark über die Bedienung der App gesprochen, was gut funktioniert hat, was nicht so gut funktioniert hat, jetzt ist für mich auch nochmal interessant, wenn man jetzt die Unterrichtsstunde an sich anschaut und reflektiert, was methodisch und didaktisch, didaktischer Aspekte angeht. Was hätte Sie da für Feedback, was hat denn ihrer Ansicht nach gut funktioniert, wo hat man gemerkt, dass da noch Möglichkeiten sind, Stellschrauben sind, die man nachziehen könnte, um noch einen besseren Effekt zu bekommen?

T: Also einmal kurz zum Methodischen, da ist mir gerade eingefallen, dass es auf jeden Fall sehr gewinnbringend, dass man auch ganz verschiedene Methoden nutzen kann. Auch die Sozialform, dass die Kinder alleine arbeiten könne, zu zwei arbeiten können, auch in Gruppen gerade wenn man vielleicht nicht so viele Geräte hat ist es natürlich auch möglich dann in Gruppen zu arbeiten und die Aufträge zu

verteilen. Der eine darf sprechen, der nächste in der nächsten Runde und so weiter, dass man das aufteilt und aus der ja eher didaktischen Sicht, was ich vorhin schon angesprochen habe, dass es exemplarisch für andere Themen nutzbar ist, das ist für uns natürlich ein ganz wichtiger Punkt, dass man sich vorstellen kann nicht nur Lernprozesse in diesem geometrischen Bereich sondern auch in ganz anderen Bereichen nicht nur im Matheunterricht sondern auch für andere Fächer zu nutzen und didaktisch gesehen ist eben auch der Punkt das das Thema für Kinder sehr zugänglich ist. Dass die Kinder jeden Tag mit Medien ja, in Berührung kommen. Die meisten Grundschüler der 4. Klassen haben selber Smartphones, die sie in der Schule meistens nicht benutzen dürfen, meistens nicht einmal mitbringen dürfen und deswegen spielt das Thema Smartphonnutzung und mit Smartphones lernen, Apps nutzen eine ganz große Rolle für die Kinder und gerade ja, diese Gegenwartsbedeutung, die dann natürlich aber auch die Zukunftsbedeutung mit sich zieht, dass es einfach ja wichtig ist jetzt auch schon Grundlagen zu lernen für das weitere Lernen, das ist für mich ein ganz wichtiger Punkt. Und was ich mir noch gut vorstellen könnte, was natürlich jetzt ganz andere Anforderungen wahrscheinlich an dieser App stellt ist jetzt ein Punkt, der quasi sehr wenig jetzt beachtet wurde, nämlich der Bereich Differenzierung. Dass ja gerade wenn Kinder unterschiedliche Sprachmöglichkeiten haben oder verschieden schnell sprechen, unterschiedlich sprechen, manche können vielleicht nicht so sprechen, dass die App das aufnehmen kann, in welcher Art auch immer, dass man da ja irgendwelche Möglichkeiten bieten könnte, um das stärker zu differenzieren. Kann natürlich auch methodisch geschehen, dass man die Kinder schon anderes gruppiert um dann mit so einer App zu arbeiten.

K: Ja Sie haben gerade die Wichtigkeit angesprochen der Medien der digitalen Technologien auch in der Schule. Welcher Einsatz kommt denn so im alltäglichen, ja Unterrichtsgeschehen zu tragen. Was setzten Sie denn ja, täglich ein

oder wöchentlich? Gibt es da Möglichkeiten digitale Medien zu nutzen, um in den verschiedensten Unterrichtsfächern dort methodisch zu unterstützen oder wird das im Moment als, speziell an Ihrer Schule, nicht ganz so im Fokus, in den Fokus gerückt?

T: Ja das ist auf jeden Fall auch von Schule zu Schule ganz unterschiedlich. Bei uns an der Schule gibt es keine Smartboards, wir haben in den Klassenräumen selber eine Tafel und haben die Möglichkeit da eine Overheadprojektor reinzuschieben, einen Fernseher, wir haben einen Beamer den wir reinstellen können, um dann an den Wänden mit dem Beamer zu arbeiten, haben aber ja keine Smartboards und auch keinen ganzen Satz Tablets für eine Klasse. Es gibt ja auch Schulen mit Tabletklassen zum Beispiel. Wir haben einen recht gut ausgestatteten Computerraum wo verschiedene Geräte zur Verfügung sind. Unter anderem verschiedene Laptops, dann Computer und eben auch wieder ein Beamer für alle und haben auch ein Smartboards für die Schule vorhanden, an dem wir arbeiten können was natürlich in dem Fall wieder mit einem Raumwechsel verbunden ist und dann ja, auch negative Aspekte mit sich bringt, wenn man wieder erst mehr Zeit braucht, Organisation, andere Vorbereitung und so weiter. Da gibt es hingegen ganz andere Schulen, die in jedem Raum oder vielen Räumen Smartboards und damit dann arbeiten können.

K: Ok. Das heißt sie haben jetzt nicht so den, so direkte Erfahrung mit so einer Tabletklasse und ob das sinnvoll oder nicht, aber haben Sie da ein Gefühl für, eine Meinung zu dem Einsatz von Tablets, Smartphones immer stärker in den Unterricht zu integrieren? Vorteile, Nachteile?

T: Ja natürlich. Also ich habe da eine ganz klare Meinung dazu, bin auf jeden Fall dem gegenüber positiv eingestellt und würde das gerne viel öfter machen, am liebsten täglich da, ja gibt es natürlich Punkte, die dagegen sprechen warum wir es an der Schule nicht machen können und gerade dass man eben dann ja, keine geeignet Ausstattung dafür hat, wo dann auch hohe Kosten einfach auch damit verbun-

den wären um das anzuschaffen. Das spricht bei uns eben dagegen und ja auf der Seite der negativen Punkte sehe ich auch manchmal den Zeitfaktor, dass man immer sagt, man muss das und das im Unterricht schaffen im Kerncurriculum stehen Punkte die abgearbeitet werden müssen, quasi und dadurch so etwas einfach ja wieder mehr Zeit in Anspruch nimmt, das manche sagen deswegen haben wir die Geräte an manchen Schulen, nutzen sie aber selten. Und ja für gerade für unsere Schule, wo es solche Geräte jetzt nicht für jede Klasse gibt oder auch keine Tabletklasse wäre es eben auch wieder ein negativer Punkt wenn man sich dann einarbeiten müsste, das Lehrer einfach nicht gut genug vorbereitet sind, um mit ja Klassen mit Tablets zu arbeiten weil das in der Ausbildung nicht vorhanden ist und dadurch wäre einfach die Zeit für die Einarbeitung und Vorbereitung dann auch wieder so hoch, dass sich, ja das wahrscheinlich am Ende nicht richtig lohnen würde. Was für mich aber ganz ganz klar dafür spricht ist, dass es ja ein wichtiges Werkzeug ist mit dem man viel Lernen kann Lernprozesse gefördert werden können, wie wir jetzt heute auch gesehen haben ist es sehr sehr motivierend, die Kinder können kooperativ arbeiten, also auch wieder diese ganzen sozialen Kompetenzen werden angesprochen. Für mich ist noch ein weiterer wichtiger Punkt, dass Kinder auch selbstständig damit umgehen können. Sie sollen ja heutzutage immer mehr in ihrer Selbstständigkeit gefördert werden, sei es, dass sie selber sich die verschiedenen Arbeitsaufträge holen und das ist an dieser Stelle auch anzumerken, dass Kinder durch Mediennutzung auch wieder viel in ihrer Selbstständigkeit gefördert werden können und dann eben auch besser darauf vorbereiten würden, wenn sie in der weiterführenden Schule mit Geräten arbeiten müssten.

K: Ok. Ja dann erst mal auf jeden Fall vielen Dank, dass Sie sich die Zeit genommen haben und wir haben ja schon besprochen, ich werde jetzt die Fragebögen auswerten und dann können wir nochmal zusammen auf die ja Antworten der

Schüler schauen und dann würde ich mich freuen wenn ich dann einmal noch einmal eine Einschätzung von Ihnen bekomme, wieso vielleicht eine bestimmte Antwort eines Schülers gegeben wurde. Da haben Sie ja nochmal einen bisschen besseren Blick vor allem, weil sie auch die Kinder ja, sehr gut einschätzen können, wenn Sie mit denen die ganze Zeit arbeiten.

T: Ja, gerne.

C.2. Gruppenarbeit 1 Transkript (G1)

Pfad zur Datei (Datenträger)	Aufnahmen/AufnahmeGruppenarbeit1.mp3
Zeitstempel	27.11.2017 12:30
Ort	Poggenhagen
Dauer	19 Minuten, 27 Sekunden
Datenerhebung	Gruppenarbeit
Teilnehmer/in	Anonymisierte/r Schüler/in 1 (S1) Anonymisierte/r Schüler/in 2 (S2) Marcel Kaufmann (K) Sonja Teichmann (T)
Transkriptionsregeln	Die Transkription erfolgte Wort für Wort ohne Sprachglättung. Die Regeln sind in Abschnitt 7.2 detailliert dargestellt.
Verwendete Transkriptionsmodule	- Sprachglättung - Pause - Sprachklang - Zeichensetzung - Nicht-sprachliche Ereignisse - Unsicherheit, Unterbrechung, Auslassung - Zeichensetzung - Interaktion

```
1  S1: Jon. J, O, N.
2  S2: Was?
3  S1: J, O, N.
4  S2: J?
5  S1: Ja!
6  S2: Ohja.
7  S1: Oh du weißt, (...?) oh oh oh
8  S2: Jon.
9  S1: (lacht) --- ⌊ es läuft.
10 S2:             ⌊ (...?) Jor...
11 S1: genson
12     [...] #00:00:22 bis 00:00:36#
13 S2: Jon Jorgenson. Speichern.
14 S1: Du musst doch auf ⌊ Start...
```

S2: ⌊ Jon Jorgenson. Start. (lacht)
S1: So ok jetzt müsste das hier durchlesen. Lösche alle Fakten, die envent⌊uell noch vorm vorherigen Körper...
S2: ⌊Ja jetzt weiß ichs jetzt weiß ichs. Wie viele Ecken hat ein, wie viele
S1: Ecken, du weißt nicht was Ecken sind?
S2: Nein ich meine eigentlich wenn...
S1: Eins, zwei, drei, vier, fünf, sechs, sieben, acht.
[...] #00:01:06 bis 00:01:25#
S2: Ein Quader hat acht Ecken.
K: Genau, also mach mal: Genau also ein Quader hat acht Ecken heißt ja dann, dass das für alles gilt.
S1: Hah, ich habs dir gesagt, ich habs dir gesagt.
S2: Ich sehe acht Ecken.
K: Genau oder ihr könnt genauso gut sagen: Da sind, oder so.
S2: Ok.
S1: Sechs, ich sehe sechs Flächen, darf i:::ch?
S2: Ich sehe sechs Flächen.
S1: Warum hast du (eigentlich?) (alle?)
S2: Jetzt darfst du.
S1: Ja::: genau. Kanten. Eins, zwei ....
S2: Wie viele sinds?
S1: Keine Ahnung, eins, zwei
S2: Warte, soll ich mal g... -- Eins, zwei, drei, vier, fünf, sechs, sieben, acht. - Kanten.
S1: Ich sehe acht Kanten. --- Ja, sehr gut, was jetzt? Wie war das?
[...] #00:02:46 bis 00:03:09#
S1: So und was jetzt? Wie war das?
S2: Les dir den Zettel durch.
(Pause)
S2: Füge die Regeln hinzu und (...?)
S1: Das ham wir schon.
S2: Frage, ob (sede?) mit dem gege, gegebenen ⌊ gegebenen
S1: ⌊ Nene, wir sind jetzt hier bei Aufgabe 3. -- Füge...
S2: (...?) Regel hinzu die beschreibt was die Fakten auf die

ein Quader schließen.
S1: So soll ich jetzt sagen das ist ein Quader? -- Das ist ein Quader. Das ist ein Quader.
(Pause)
S2: Soll ich nochmal, | das ist...
S1: | Die hat irgendwas gesagt.
S2: Das ist ein Quader. -- Geht doch, Quader. --- Aber da kommt nichts.
S1: Pech. --- Äh nein, doch nicht Quader. ---
S2: Doch, Quader.
[...] #00:04:18 bis 00:05:23#
T: Gut, dann könnt ihr jetzt sagen, wenn es das und das und das hat, ist es ein...
S2: Ahhh.
S1: Quader
T: Dann los.
S2: Wenn es acht Kanten, sechs Flächen und acht Ecken hat, ist es ein Quader. --- Wä Wä (lacht)
S1: (lacht+) Wir habens geschafft.
S2: Aber das vordere muss ich löschen.
S1: Was? Nein!
S2: Doch das steht doch dadrauf.
S1: Wieso? --- (singend+) Wir hams geschafft!
S2: (singend+) Wir hams geschafft!
[...] #00:05:41 bis 00:06:05#
K: Was habt ihr schon fertig?
S2: Wir haben, was ham wir fertig!
S1: Quader!
(Pause)
K: Und dann, dürft ihr das austauschen, genau.
[...] #00:06:20 bis 00:06:35#
S2: Müssen wir jetzt das dadrunter machen? -- Wir nehm beide, danke.
S1: Nein wir es gibt, muss da Sachen auf ab, es muss da Sach.. die hier der Würfel muss weg es muss da eine Sache geben die zum austauschen für die anderen. Die| muss weg nach...
S2: |(Muss?) (man?)

S1: Die muss weg, die anderen müssen doch auch noch austau-
    schen. --
S2: Ok. --
S1: Hä? Wie komm wa denn jetzt... Ich frag mal kurz was
    [...] #00:07:12 bis 00:07:46#
S1: Warte. Du (mi?) mit (c?). Warte, lö⌊sche die
S2:   ⌊Also was machen wir
    jetzt zuerst? Zuerst Ecken⌋, die Ecken sind die halt das
    hier.
S1:   ⌊Ja Ecken
S1: Ja.
S2: Eine Ecke. Also das hier oben ist ja die Ecken.
S1: Ich sehe eine Ecke, ne? Ich sehe eine Ecke. ---
S2: Du hast, musst loslassen.
S1: Ich sehe eine Ecke. -- Dankeschön.
S2: So darf ich mal? Jetzt bin ich dran.
S1: Ne:::in! Du hast auch zweimal.
S2: Ok.
S1: Flächen.
S2: Zwei.
S1: Eins.
S2: Zwei.
S1: Echt?
S2: Ja zwei, ei:::ns und zwei.
S1: Stimmt. - Ich sehe zwei Flächen. -- Oh und jetze...
S2: Jetzt bin ich dran.
    [...] #00:08:40 bis 00:08:49#
S1: Los, wie viele K... Kanten hat es. ⌊Eine
S2:   ⌊Eine
S1: Ich sehe eine Kante.
S2: Ich sehe eine Kante.
S1: Oh man du hattest dreimal. Jetzt hattest du vielleicht
    fünfmal hintereinander.
S2: Du hast doch auch was gemacht. Danach darfst du die ganze
    machen.
S1: Ja danke. ---
S2: (...??) ist es ein Kegel.

S1: Ja!
S2: Ja!, drittes
    [...] #00:09:24 bis 00:09:48#
 T: Jetzt kann getauscht werden.
S2: So her ⌊damit
S1:   ⌊So jetzt bin ich dran.
S2: Hier. -- Ich zähle, wir sind bei.
S1: Pyramide (...??) So. Die Pyramide hat...
S2: Öhm
S1: Fünf Kanten? Ne? -- Zähl! -- (Oder mach lieber die?)
    Ecken, Ecken.
S2: Eins, zwei, drei, vier, fünf.
S1: Ecken sind das hier.
S2: Eins, zwei, drei, vier, fünf.
S1: Ich sehe fünf Ecken. --
S2: Eins, zwei, drei, vier, fünf Flächen.
S1: Ich sehe fünf Flächen. -- Und jetze Kanten ne?
S2: Also sind das hier die langen? Eins, zwei, drei, vier,
    fünf, sechs, acht.
S1: Ich sehe acht -- Kanten.
S2: Ok, warte mal.
    [...] #00:10:58 bis 00:19:27#

C.3. Gruppenarbeit 2 Transkript (G2)

Pfad zur Datei (Datenträger)	Aufnahmen/AufnahmeGruppenarbeit2.mp3
Zeitstempel	27.11.2017 12:30
Ort	Poggenhagen
Dauer	17 Minuten, 58 Sekunden
Datenerhebung	Gruppenarbeit
Teilnehmer/in	Anonymisierte/r Schüler/in 1 (S1) Anonymisierte/r Schüler/in 2 (S2) Marcel Kaufmann (K) Sonja Teichmann (T)
Transkriptionsregeln	Die Transkription erfolgte Wort für Wort ohne Sprachglättung. Die Regeln sind in Abschnitt 7.2 detailliert dargestellt.
Verwendete Transkriptionsmodule	- Sprachglättung - Pause - Sprachklang - Zeichensetzung - Nicht-sprachliche Ereignisse - Unsicherheit, Unterbrechung, Auslassung - Zeichensetzung - Interaktion

```
1  S1: Also. Äh Name der (...??)
2      (Pause)
3  S1: Achso da sollen wir irgendwas...
4  S2: (...?) ist meine Mini-Welt
5  S1: Hallo, das ham wir schon gemacht, wir sind da ---
6  S2: Ok, von, ähm das hier das ähm hat, es hat zwei - Kanten
7      (...?) Kanten.
8  S1: Es hat zwei Kanten. --
9  S2: Hallo es klappt nicht.
10 S1: Könnt ihr mal leise sein!
11 S2: Es hat zwei Kanten. --
12 S1: Die sind zu laut.
13 S2: Es , ne wir müssen Flächen (...?) --
14     [...] #00:01:38 bis 00:01:51#
```

S2: Das nimmt, das nimmt nicht auf. Damit sind wir (...?)
Jetzt sind wir auf Start gegangen jetzt müssen wir fortfahren.
T: Genau, jetzt kanns losgehen mit eurem Körper. jetzt guckt ihr, wie viele Ecken hat der.
S1: Zwei.
T: Ecken.
S1: Achso, null.
T: Gut, dann sprecht rein, ich sehe null.
S1: Ich sehe null Ecken.
S2: ⎣Nur Ecken
S1: ⎣Nur Ecken
T: Nu:::r hat er verstanden du musst nu:::ll ⎣sagen.
S2: ⎣Lass mich mal -- Ich sehe null Ecken.
S1: Geht doch. Und wie viele Kanten?
S2: (...?) ---
S1: Ich sehe zwei Kanten. Verstehst dus jetzt? Da:::nke. Fei:::n das. Alter voll das geile (...?) Handy. Ich will auch so eins.
S2: Eine Fläche, eine Fläche ne?
S1: Drei::: Eins, zwei, drei
S2: (...?) drei Flächen. -- Ich sehe drei Flächen. (...?)
S1: Regel.
S2: *Regeln*...
S1: Wenn es null Ecken, zwei Kanten und warte - Wenn es null Ecken, zwei Kanten und drei Flächen hat, ist es ein Zylinder. - Nur Ecken? Nicht schon wieder diese nur Ecken.
T: Äh hier könnt ihr löschen bei dem Mülleimer. Genau, dann macht ihrs nochmal.
S2: Wenn es
S1: Wenn es nu:::ll
S2: Ecken
S1: Ecken zwei Kanten und drei Flächen hat, ist es ein Zylinder. --- Ich sag doch ist ein dummes Handy.
T: Hat er noch nicht aufgenommen?
S1: Nein.

S2: Lass mich mal.
T: Habt ihr es laut genug gemacht?
(Pause)
S2: Wenn es null Ecken, zwei Kanten und drei Flächen hat, ist es ein Zylinder. --
S1: Bringt nichts (...?)
T: Euer App... Ist abgestürzt.
S1: Wieso ist es abgestürzt?
T: Jetzt macht ihrs nochmal und dann müsste es auch klappen. --
S2: Wenn es null Ecken, zwei Kanten und drei Flächen hat, ist ein Zylinder. --
K: Versucht...
S2: Wenn es null Ecken, zwei Kanten und drei Flächen hat, ist ein Zylinder. --
K: Hat es denn, Hat es denn geplinkt als du drauf gedrückt hast?
S2: Ja. Es hat ja eben grad schon wieder geplinkt. ---
K: Ähm, --- versuch mal mit keine Ecken. Vielleicht ähm, kann er das noch nicht.
S2: Wenn es keine Ecken zwei Kanten und drei Flächen sind, ist es ein Zylinder.
(Pause)
S1: Wenn es keine Ecken hat, zwei Kanten und drei Flächen, ist es ein Zylinder. Ye⌊ah
S2: ⌊Danke!
S1: Hä? Zwei? Kann mir mal jemand sagen warum da jetzt zwei steht?
(Pause)
S2: Wenn es keine Ecken, zwei Kanten und drei Flächen hat, ist es ein Zylinder. --- Willst du nochmal?
S1: Wenn es keine Ecken hat, zwei Kanten und drei Flächen, ist es ein Zylinder.
(Pause)
S2: Wenn es keine Ecken, zwei Kanten und drei Flächen hat, ist es ein Zylinder.
S1: Fertig.

S2: Frag ob SLIDE (...??)
S1: (...??) mit den gegebenen Fakten (...??) Lass dir vom Lehrer die Form auf dem Laufzettel abhaken.
(Pause)
S2: Frag ob SLIDE mit den ge gegebenen Fakten dir bestätigen kann, dass es sich um ein Zylinder handelt. Lass dir vom Lehrer die (...?) auf dem Laufzettel abhaken.
S1: Ok.
S2: Ist es ein Zylinder?
(Ja, das trifft zu.)
S2: Ja, wir haben ihm was beigebracht.
S1: Ok, nochmal. (räuspern) Ha:::llo::: ist es ein Zylinda:::. – Ha! Ich hab sogar hallo gesagt.
T: Ein Teambild.
S2: Yeah.
T: Super.
S2: Ist es ein Zylinder?
(...??)
S1: Ist es ein Zylinder?
(...??)
S2: Wir sind fertig mit dem |(...?)
S1: |Liebes Handy ist es ein Zylinda:::?
(Ja, das trifft zu.)
S1: Hah, es funktion|iert.
T: |Ihr seid richtig gut. Dann dürft ihr schon hier tauschen. Jetzt kann getauscht werden.
S1: Ach da mussten Sie kurz noch den Laufzettel abhaken.
T: Ja ihr könnt das anmalen oder abhaken.
[...] #00:09:20 bis 00:09:44#
S1: Was? Der Kegel. Ok wir müssen alles löschen bis auf diese (...??) bis auf diese Regel. (...??)
S2: Lösche alle Fakten, die eventuell noch von den vorherigen Körper vorhanden sind, behalte aber die Regel. Löschen jetzt? (...??)
S1: Du bist doof. Toll jetzt hast du...
S2: Ist doch immer noch da. Draufklicken, runter. Ist doch im-

mernoch alles da. Ich weiß gar nicht was du hast. Guck ist doch noch⌊da.

S1: ⌊Ja ich weiß das, bin ja nicht dumm. Ok.

(Pause)

S1: *Ich sehe eins zwei drei vier fünf*

S2: Sechs.

S1: Ich sehe fünf Ecken

S2: Das ist falsch.

[...] #00:10:13 bis 00:10:36#

S1: Kegel. Wie viel. Eine Kante ---

S2: Ich sehe eine Kante.

S1: (...?) super (...?).

S2: Das sind doch auch Kanten.

S?: Das ⌊sind Kanten

S?: ⌊Ecken

S?: Ich sehe eine

S1: Ich sehe eine Ecke.

S2: Ich sehe (...?) das ist die Ecke.

S1: Das ist die Ecke. Warte.

S2: Ist doch egal nein ist doch richtig, ist doch richtig.

Guck mal eine Ecke, zwei Ecken. -- Ich sehe eine - Kante.

(...?) Ich sehe eine Fläche.

S1: Zwei.

S2: Eine (guck?)

S1: Eins, zwei.

S2: Ne, du hast stimmt. - Ich sehe zwei Flächen. -- Ist es ein

S1: Warte. Wenn es eine Ecke eine Kante und zwei Flächen hat

ist es ein Kegel.

S2: Ja::: wir ham. Jetzt bin ich nochmal. (...?) Ist es ein -

Kegel? ---

S1: (räuspern) Ist es ein Kegel?

(Ja, das trifft zu.)

S2: (...??) was neues.

[...] #00:12:24 bis 00:12:42#

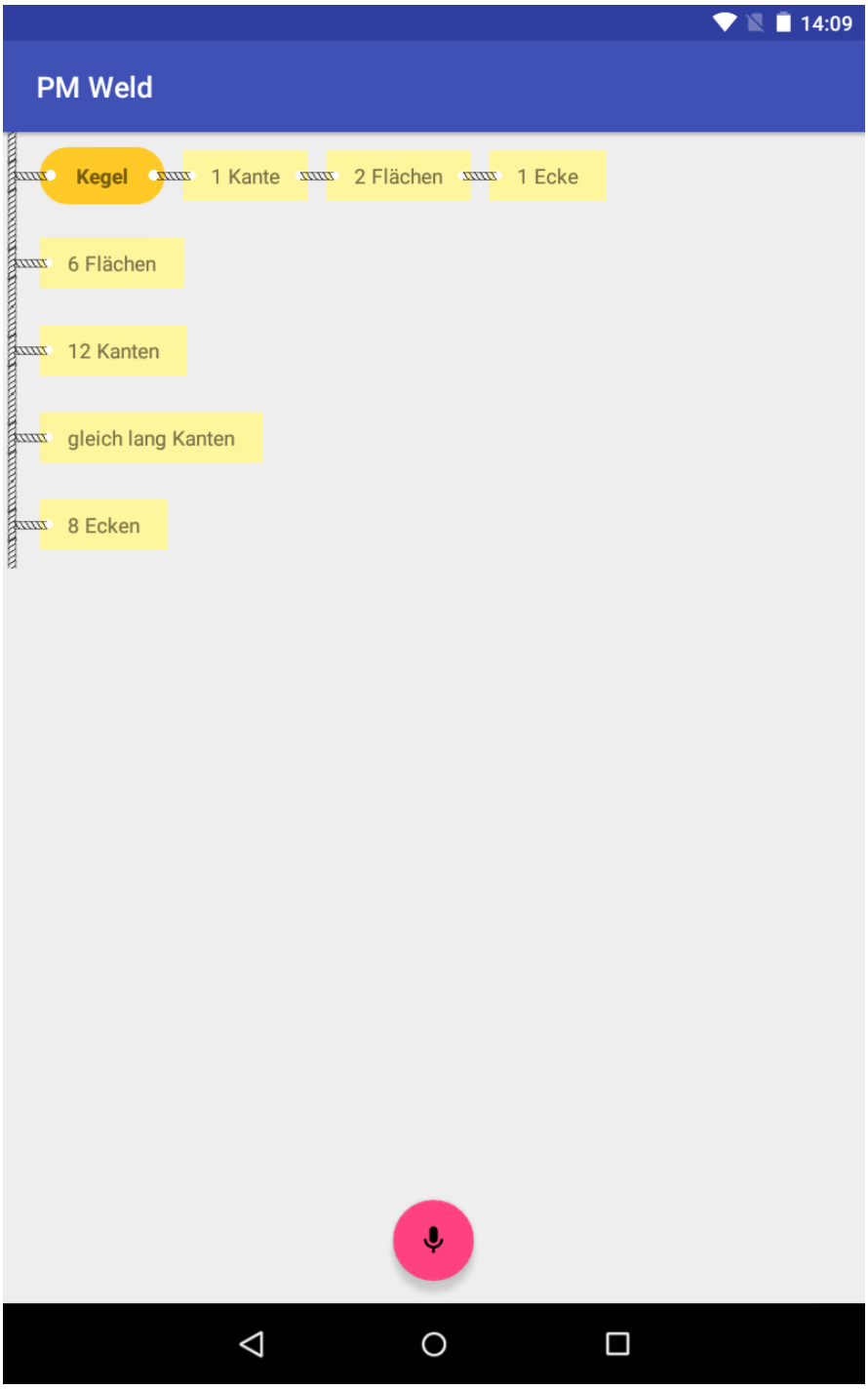
S1: Also wie viele Ecken und Kanten du zählst immer, ich sage.

S2: Eins, z⌊wei, drei

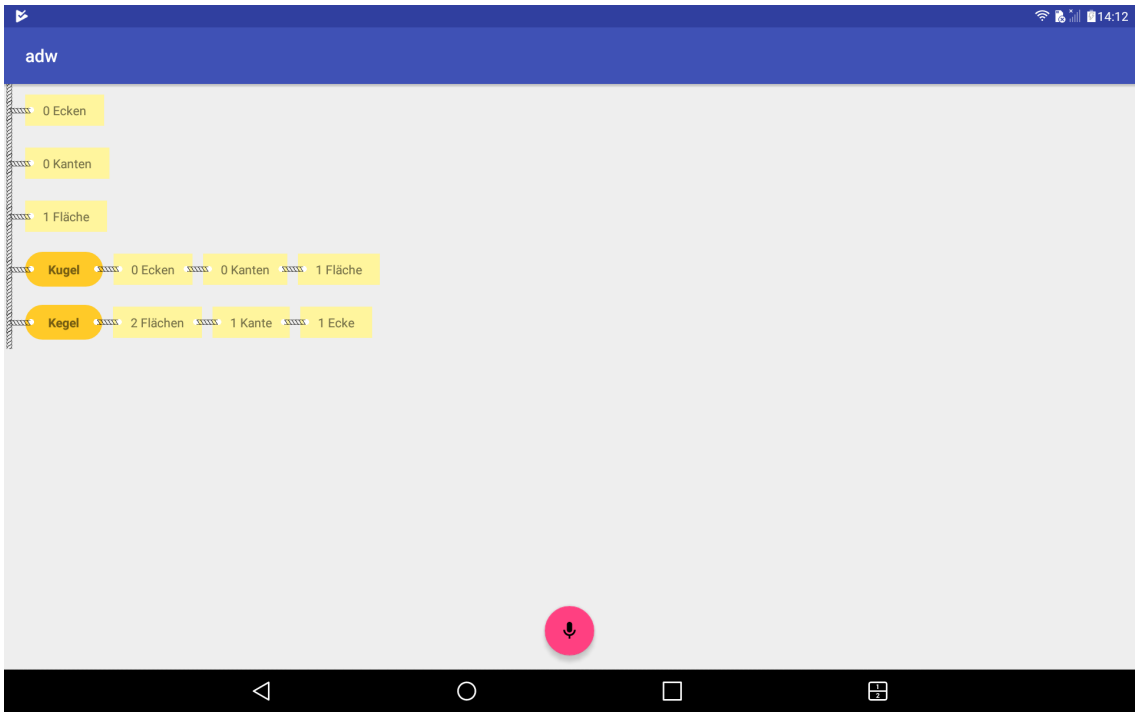
S1: ⌊Ecken, Ecken, Ecken.

S2: Ich sehe eins, zwei, dr... Achso Ecken. Das hier ne. Ich sehe eins |zwei drei vier fünf sechs sieben acht.
S1: |Acht.
S1: Ich sehe acht Ecken. ---
S2: Eins zwei drei, warte, eins, zwei, drei, vier, fünf, sechs. Sechs Flächen
S1: (...?) nochmal. --- Ok nochmal. (...?) eins, zwei, drei, vier, fünf, sechs, gut.
S2: Ich sehe sechs Flächen. ---
S1: Eins, zwei, drei, vier, fünf, sechs, sieben, acht, nein, zehn, elf, zwölf. Zwölf Kanten. Ich sehe zwölf Kanten. -- Geht doch.
S2: Wenn es acht Kanten, sechs Flächen und zwölf Kanten sind, dann ist es ein Würfel. -- Warum, warum (...??)
S1: Nein das ist falsch, das ist falsch (...??)
S2: Wenn es acht Kanten, sechs Flächen und zwölf Kanten (...??) Würfel.
(Pause)
S2: Wenn es acht Ecken, sechs Flächen und zwölf Kanten hat, ist es ein Würfel.
[...] #00:14:30 bis 00:15:39#
S2: Ich sehe eine Fläche. ---
S1: (...??)
S2: Ich sehe eine Fläche. Wenn es eine Fläche ist, dann ist es ein Würfel. --- Oh ne:::, ne Kugel. --- Wenn es eine Fläche ist, dann ist es eine Kugel.
[...] #00:16:10 bis 00:17:58#

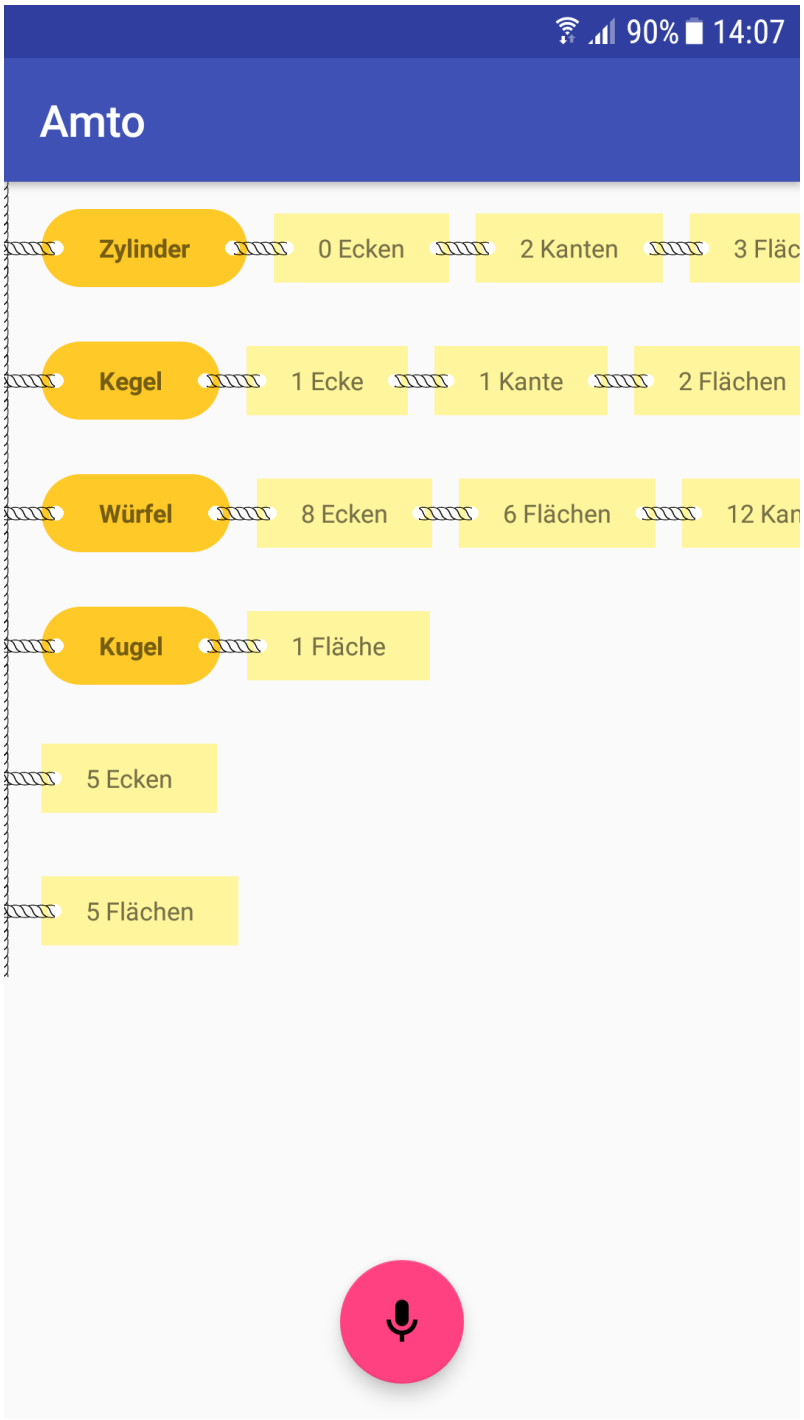
C.4. Screenshot Ergebnis Gruppe A



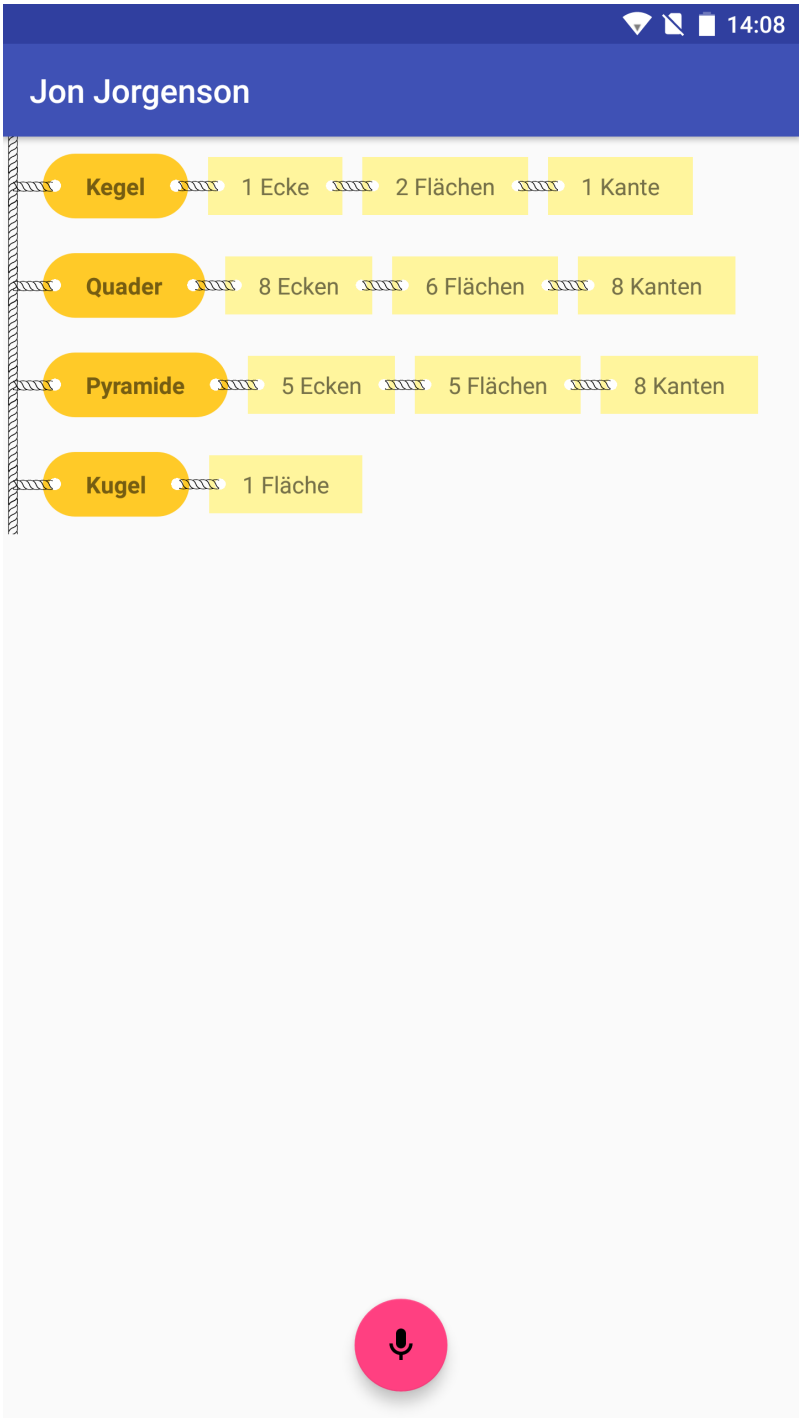
C.5. Screenshot Ergebnis Gruppe B



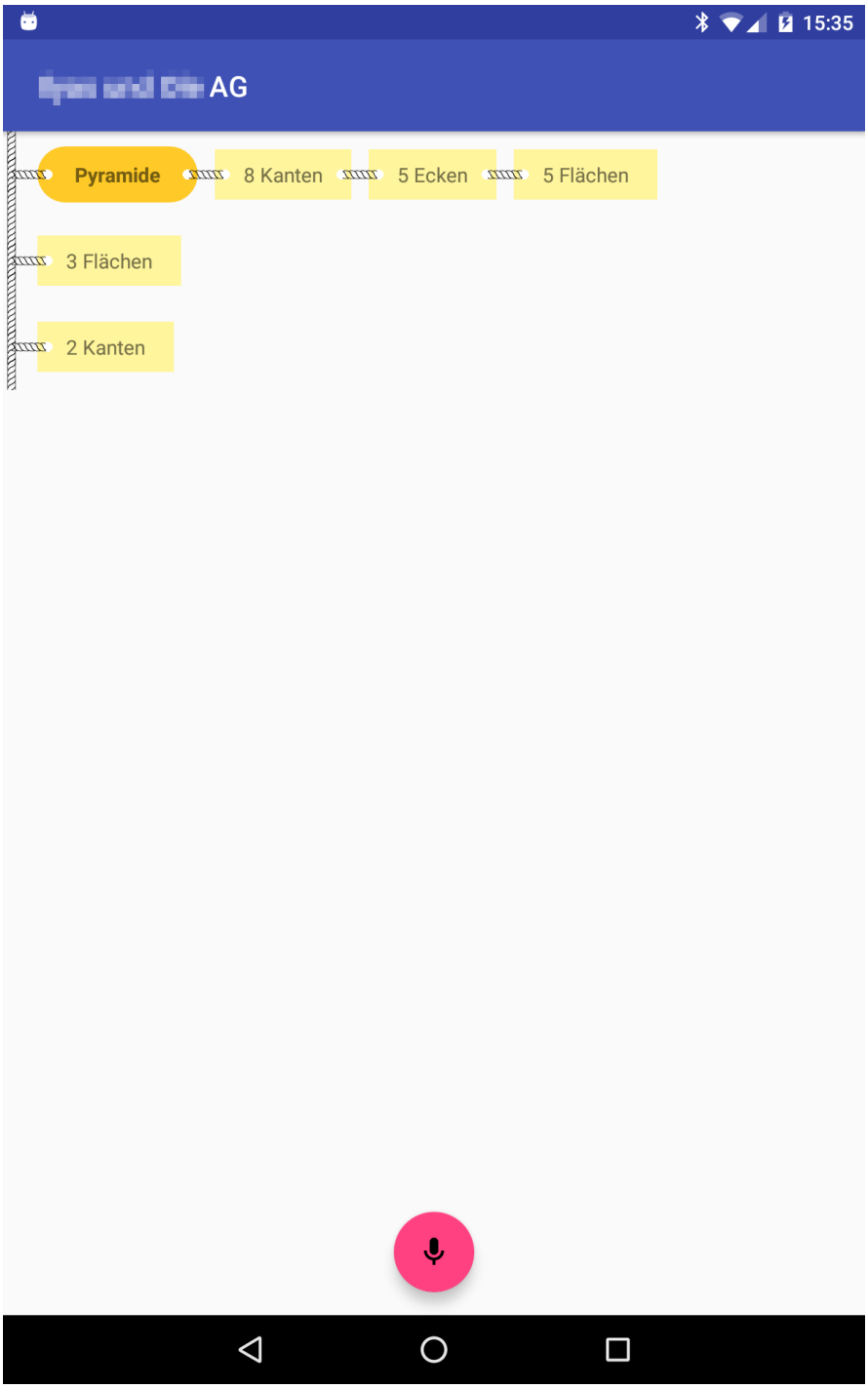
C.6. Screenshot Ergebnis Gruppe C



C.7. Screenshot Ergebnis Gruppe D



C.8. Screenshot Ergebnis Gruppe E



C.9. Auswertungsübersicht Sozialforschung

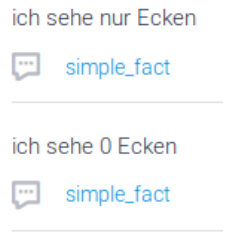
	Kategorie	Ausprägung	Beschreibung	Beobachtung

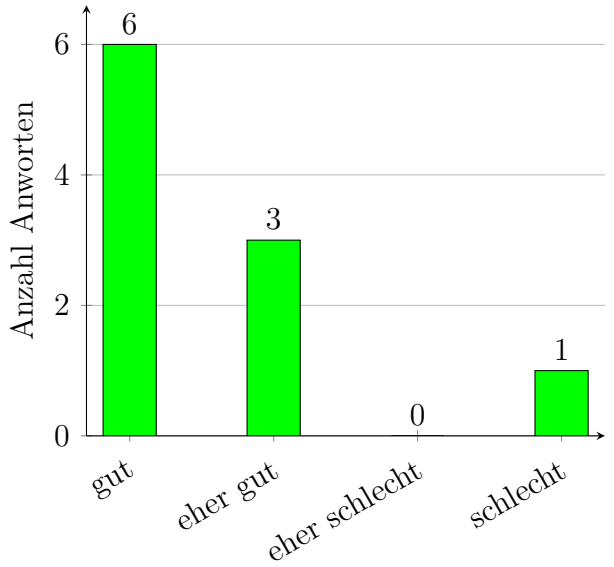
1a	Intentionen erkennen	Größtenteils fehlerfrei	Nahezu alle Intentionen wurden korrekt interpretiert	**Dialogflow Log:** *ist es eine Pyramide –> ask_query* (siehe dialogflowHistory Anhang D.3 Zeile 26), *wenn es keine Ecke zwei Kanten und drei Flächen sind ist es ein Zylinder –> common_rule* (siehe dialogflowHistory Anhang D.3 Zeile 98), *ist es nun eine Kugel ja oder nein –> ask_query*(siehe dialogflowHistory Anhang D.3 Zeile 76)
				G2(Zeile 97-99):
				`S2: Ist es ein Zylinder?`
				`(Ja, das trifft zu.)`
				`S2: Ja, wir haben ihm was beigebracht.`

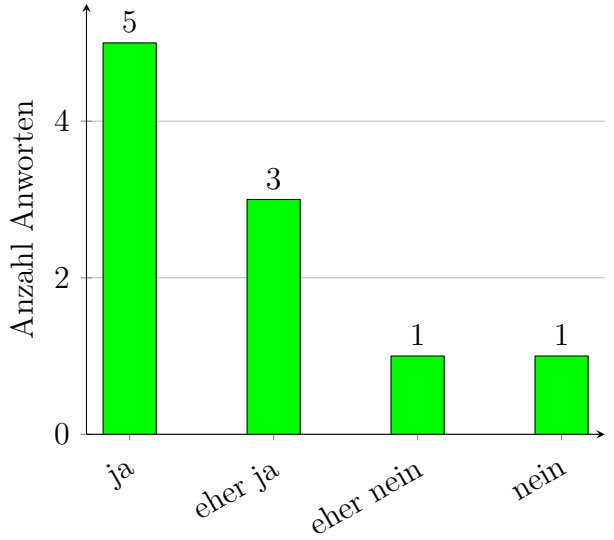
		Teilweise fehlerfrei	Intentionen wurden meistens korrekt interpretiert	**Dialogflow Log:** *liebes Handy ist das ein Zylinder –> simple_fact* sowie: *liebes Handy ist es ein Zylinder –> ask_query* (siehe dialogflowHistory Anhang D.3 Zeile 107-108)
		Größtenteils fehlerhaft	Intentionen wurden selten korrekt interpretiert	**Dialogflow Log:** *Laura bandero –> common_rule* (siehe dialogflowHistory Anhang D.3 Zeile 18)

	Kategorie	Ausprägung	Beschreibung	Beobachtung

1b	Entitäten erkennen	Größtenteils fehlerfrei	Nahezu alle Entitäten wurden korrekt interpretiert	**G2(Zeile 97-100):**
				`S2: Ist es ein Zylinder?`
				`(Ja, das trifft zu.)`
				`S2: Ja, wir haben ihm was beigebracht.`
				`S1: Ok, nochmal. (räuspern) Ha:::llo::: ist es ein Zylinda:::. - Ha! Ich hab sogar hallo gesagt.`
				
				Dialogflow Log: *ich sehe 2 Flächen –> simple_fact* (siehe dialogflowHistory Anhang D.3 Zeile 115)

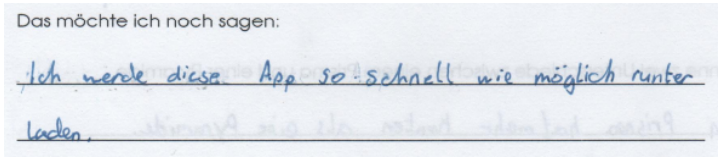
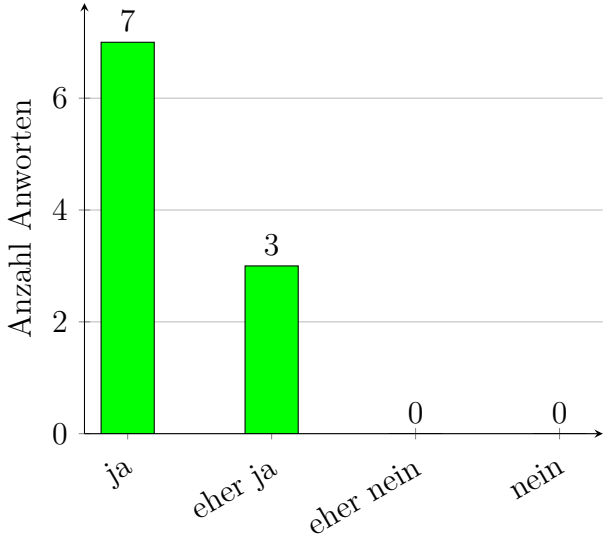
		Teilweise fehlerfrei	Entitäten wurden meistens korrekt interpretiert	**Dialogflow Historie:**  **G2(Zeile 40):** `S1: Wenn es null Ecken, zwei Kanten und warte - Wenn es null Ecken, zwei Kanten und drei Flächen hat, ist es ein Zylinder. - Nur Ecken? Nicht schon wieder diese nur Ecken.`

	Kategorie	Ausprägung	Beschreibung	Beobachtung
		Größtenteils fehlerhaft	Entitäten wurden selten korrekt interpretiert	
2a	Verständnis App Bedienung	Keine Bedienprobleme	Es war zu jederzeit klar, wie die App zu bedienen ist	**Fragebogen:**  Ich bin gut mit der App zurecht gekommen

	Kategorie	Ausprägung	Beschreibung	Beobachtung
				 Ich wusste immer, was ich drücken musste **G1(Zeile 10-15):** S2: ⌊ (...?) Jor... S1: genson [...] #00:00:22 bis 00:00:36# S2: Jon Jorgenson. Speichern. S1: Du musst doch auf ⌊ Start... S2: ⌊ Jon Jorgenson. Start. (lacht)

	Kategorie	Ausprägung	Beschreibung	Beobachtung
		leichte Bedienprobleme	Es war meist klar, wie die die App zu bedienen ist. Es brauchte Unterstützung	**G2(Zeile 43):** T: Äh hier könnt ihr löschen bei dem Mülleimer. Genau, dann macht ihrs nochmal. **G1(Zeile 102):** S2: Du hast, musst loslassen. **Feldbeobachtung:** Es war ersichtlich, dass manche Gruppen bei der Aufnahme den Mikrofon-Button gedrückt halten wollten, solange sie reinsprechen.
		starke Bedienprobleme	Es war nicht klar, wie die App zu bedienen ist	
2b	Motivation beim Bearbeiten	Starke Motivation	Die SuS waren die ganze Zeit stark motiviert	**E1(Zeile 123-124):** T: Also es war auf jeden Fall sehr sehr motivierend für die Kinder.

	Kategorie	Ausprägung	Beschreibung	Beobachtung
				E1(Zeile 143-147): T: [...] für die Kinder war es auch visuell einfach sehr ansprechend und durch diese Anschaulichkeit, die auch durch die klaren Strukturen dann in der App da waren und diese WOW Effekte, dass wenn man jetzt was spricht wird da gleich etwas angezeigt [...] **G1(Zeile 74-75):** S1: Wieso? --- (singend+) Wir hams geschafft! S2: (singend+) Wir hams geschafft! **G2(Zeile 97-99):** S2: Ist es ein Zylinder? (Ja, das trifft zu.) S2: Ja, wir haben ihm was beigebracht.

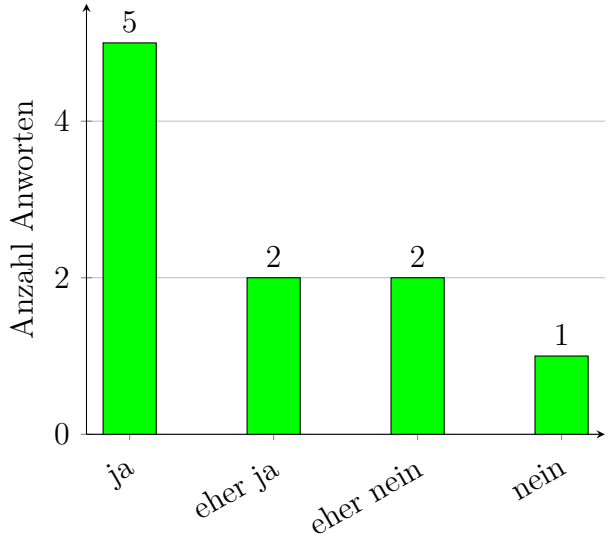
	Kategorie	Ausprägung	Beschreibung	Beobachtung
				G2(Zeile 110-113): S1: ⌊Liebes Handy ist es ein Zylinda:::? (Ja, das trifft zu.) S1: Hah, es funktion⌊iert. **Fragebogen:**   Ich fand das Lernen mit der App interessant

	Kategorie	Ausprägung	Beschreibung	Beobachtung
		Leichte Motivation	Die SuS waren teilweise motiviert und mussten teilweise durch die Lehrkraft zum Weiterarbeiten angehalten werden	**G2(Zeile 55-59):** S2: Wenn es null Ecken, zwei Kanten und drei Flächen hat, ist es ein Zylinder. -- S1: Bringt nichts (...?) T: Euer App... Ist abgestürzt. S1: Wieso ist es abgestürzt? **G2(Zeile 40-42):** S1: Wenn es null Ecken, zwei Kanten und warte - Wenn es null Ecken, zwei Kanten und drei Flächen hat, ist es ein Zylinder. - Nur Ecken? Nicht schon wieder diese nur Ecken. **G1(Zeile 58-59):** S2: Das ist ein Quader. -- Geht doch, Quader. --- Aber da kommt nichts.

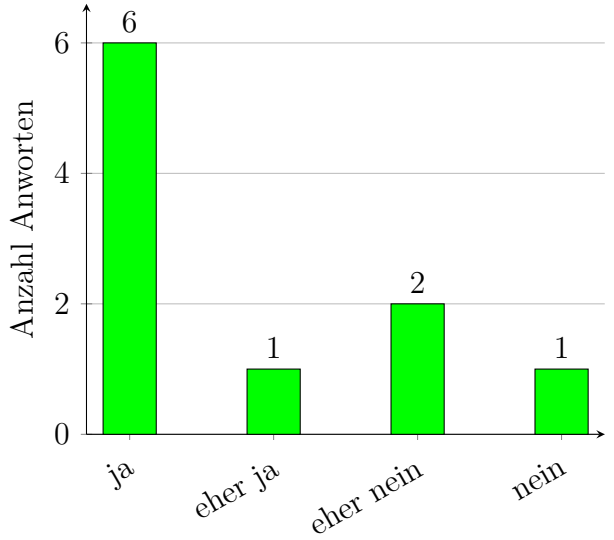
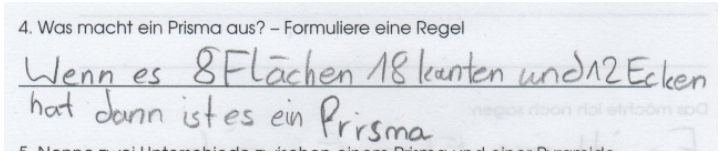
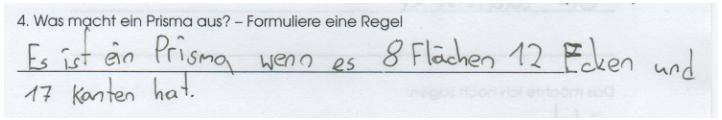
	Kategorie	Ausprägung	Beschreibung	Beobachtung
		Keine Motivation	Die SuS waren nicht motiviert und mussten immer wieder durch die Lehrkraft zum Weiterarbeiten angehalten werden	

2c	Anwenden der Fakten in gelernter Form	Korrektes Anwenden	SuS wenden ihr erworbenes Wissen ohne Hilfe an	**G1(Zeile 28):**
				`S2: Ich sehe acht Ecken.`
				G1(Zeile 32):
				`S2: Ich sehe sechs Flächen.`
				G1(Zeile 40):
				`S1: Ich sehe acht Kanten.`
				
				Fragebogen:
				9 SuS haben die Eigenschaften mit den Fachbegriffen als Fakten formuliert. Korrekt und vollständig waren 4.

		Teilweise korrektes Anwenden	SuS benötigen Hinweise der Lehrperson	**G1(Zeile 24):** `S2: Ein Quader hat acht Ecken.` **Fragebogen:**

	Kategorie	Ausprägung	Beschreibung	Beobachtung
				 Es fiel mir leicht, einzelne Fakten zu erstellen
		Kein korrektes Anwenden	SuS können Fakten nicht umsetzen	

	Kategorie	Ausprägung	Beschreibung	Beobachtung
2d	Anwenden der Regeln in gelernter Form	Korrektes Anwenden	SuS wenden ihr erworbenes Wissen ohne Hilfe an	**G2(Zeile 187-188):** S2: Wenn es eine Fläche ist, dann ist es eine Kugel. **G2(Zeile 181-182):** S2: Wenn es acht Ecken, sechs Flächen und zwölf Kanten hat, ist es ein Würfel. **G2(Zeile 152-153):** S1: Warte. Wenn es eine Ecke eine Kante und zwei Flächen hat ist es ein Kegel.
		Teilweise korrektes Anwenden	SuS benötigen Hinweise der Lehrperson	**G1(Zeile 63-69):** T: Gut, dann könnt ihr jetzt sagen, wenn es das und das und das hat, ist es ein... S2: Ahhh. S1: Quader T: Dann los. S2: Wenn es acht Kanten, sechs Flächen und acht Ecken hat, ist es ein Quader. --- Wä Wä (lacht) **Fragebogen:**

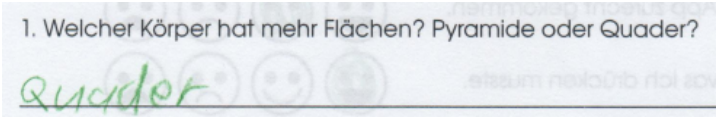
	Kategorie	Ausprägung	Beschreibung	Beobachtung
				 Es fiel mir leicht, einzelne Regeln zu erstellen 5 SuS haben die Frage, was den Prisma ausmacht, durch eine Regel in gelernter Form formuliert. Beispiele:   **Wissensbasis:**

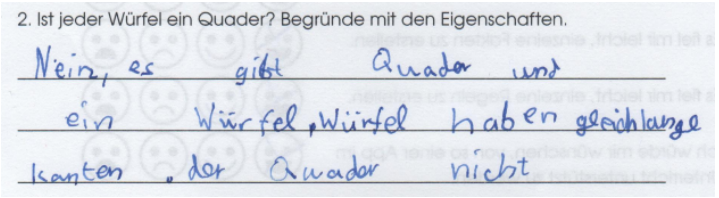
	Kategorie	Ausprägung	Beschreibung	Beobachtung
				Im Anhang C.4,C.5,C.6,C.7,C.8 sind die Ergebnisse der Wissensbasen zu sehen. Der Fortschritt ist sehr Unterschiedlich.
		Kein korrektes Anwenden	SuS können Fakten nicht umsetzen	**G1(Zeile 53-54):** S1: So soll ich jetzt sagen das ist ein Quader? -- Das ist ein Quader. Das ist ein Quader.
4a	Prozessbezogener Kompetenzbereich Argumentieren	Stark angesprochen	Argumentieren war starker Fokus beim Bearbeiten der Aufgaben	**G1(Zeile 107-113):** S1: Flächen. S2: Zwei. S1: Eins. S2: Zwei. S1: Echt? S2: Ja zwei, ei:::ns und zwei. S1: Stimmt. - Ich sehe zwei Flächen. -- Oh und jetze...

	Kategorie	Ausprägung	Beschreibung	Beobachtung
				G1(Zeile 135-140): S1: Fünf Kanten? Ne? -- Zähl! -- (Oder mach lieber die?) Ecken, Ecken. S2: Eins, zwei, drei, vier, fünf. S1: Ecken sind das hier. S2: Eins, zwei, drei, vier, fünf. S1: Ich sehe fünf Ecken. -- **G2(Zeile 147-151):** S2: Ich sehe eine Fläche. S1: Zwei. S2: Eine (guck?) S1: Eins, zwei. S2: Ne, du hast stimmt.
		Angesprochen	Argumentieren wurde beim Bearbeiten der Aufgaben benötigt	

	Kategorie	Ausprägung	Beschreibung	Beobachtung
		Nicht angesprochen	Argumentieren wurde beim Bearbeiten der Aufgaben nicht benötigt	**E1(Zeile 56-58):** T: Also hier ging es ja weniger dann darum etwas auch konkret zu belegen, zu begründen, zu hinterfragen warum etwas so ist...
4b	Prozessbezogener Kompetenzbereich Kommunizieren	Stark angesprochen	Kommunizieren war starker Fokus beim Bearbeiten der Aufgaben	**E1(Zeile 47-49):** T: Aber der Bereich Kommunizieren, der jetzt sowieso in der Schule immer ganz ganz wichtig ist, ist hier ganz stark vertreten gewesen. **E1(Zeile 134-136):** T: [...] wo eben auch wieder das Kommunizieren über etwas [...] im Vordergrund stand. **G1(Zeile 120-123):** S1: Oh man du hattest dreimal. Jetzt hattest du vielleicht fünfmal hintereinander. S2: Du hast doch auch was gemacht. Danach darfst du die ganze machen.

	Kategorie	Ausprägung	Beschreibung	Beobachtung
				G1(Zeile 104-106): S2: So darf ich mal? Jetzt bin ich dran. S1: Ne:::in! Du hast auch zweimal. S2: Ok. **G2(Zeile 35-37):** S2: Eine Fläche, eine Fläche ne? S1: Drei::: Eins, zwei, drei S2: (...?) drei Flächen. -- Ich sehe drei Flächen. (...?) **G2(Zeile 141-144):** S?: Ich sehe eine S1: Ich sehe eine Ecke. S2: Ich sehe (...?) das ist die Ecke. S1: Das ist die Ecke. Warte.
		Angesprochen	Kommunizieren wurde beim Bearbeiten der Aufgaben benötigt	

	Kategorie	Ausprägung	Beschreibung	Beobachtung
		Nicht angesprochen	Kommunizieren wurde beim Bearbeiten der Aufgaben nicht benötigt	
4c	Inhaltsbezogene Kompetenzbereiche: Mathematisches Verständnis Eigenschaften/-Unterschiede der Körper	Stark angesprochen und verstanden	Fokus waren die Eigenschaften und Unterschiede der Körper. Das Verständnis wurde stark gefördert.	**E1(Zeile 64-65):** `T: [...]gerade diese ja kausalen Zusammenhänge waren da auf jeden Fall vertreten.` **Fragebogen:** Frage 1: 9 von 10 SuS haben die Frage zu mehr Flächen (Pyramide oder Quader) korrekt beantwortet.  **Feldbeobachtung:** Die Kinder wiederholten immer wieder die Fachbegriffe Ecken, Kanten und Flächen.

	Kategorie	Ausprägung	Beschreibung	Beobachtung
		Angesprochen und teilweise verstanden	Eigenschaften und Unterschiede der Körper wurden angesprochen. Verständnisses wurde leicht gefördert.	**Fragebogen:** Frage 2: 6 von 10 SuS haben die Frage richtig beantwortet.
		Nicht angesprochen. Wenig bis nichts verstanden	Eigenschaften und Unterschiede der Körper wurden nicht angesprochen. Eine Förderung des Verständnis war nicht ersichtlich.	**Fragebogen:** Frage 2: Keine Schülerin und kein Schüler hat die Antwort korrekt begründet. Die einzige richtige Begründung eines Schülers führte zur falschen Schlussfolgerung. 

C.10. Anpassungen Fallstudie 1 Übersicht

<table>
<tr><th>Modul</th><th>Anpassung</th></tr>
<tr><td rowspan="3">domain</td><td>SpeechRecognitionHandler.kt - Ein neuer Service entsteht, damit die App STT selbst steuern kann. Dies soll über ein weiteres Handler-Interface (SpeechRecognitionHandler.kt) geschehen, das eine Service-Klasse im data-Modul implementieren kann.

<pre>interface SpeechRecognitionHandler{
 fun start(): Completable
 fun stop(): Single<String>
}</pre></td></tr>
<tr><td>Es müssen zwei neue UseCases entstehen, die das Starten und Stoppen zum Aufnehmen des Gesagten steuern. Der eine UseCase zum Start ist ein über ein Completable definiert, da dies nur einmalig und ohne spezifischen Rückgabewert ausgeführt wird. Der UseCase für Stop wird ebenfalls nur einmal ausgeführt, stellt aber im Stream das Gesagte als String zur Verfügung.</td></tr>
<tr><td>APIAIHandler.kt, ExecuteUserRequestUseCase.kt Das Gesagte in Textform muss zum Dialogflow-Service gelangen. Daher ist der entsprechende UseCase und das Interface zu erweitern. Dieses erwartet als Übergabe den String, der an den Service übergeben werden soll.

<pre>interface APIAIHandler{
 fun getApiAiResponse(input: String):
  ↪ Observable<AIResponse?>
}</pre>

<pre>return apiaiHandler⌋
 ↪ .getApiAiResponse(params.input)</pre></td></tr>
</table>

Modul	Anpassung
data	Im data-Modul entsteht eine neue Service-Klasse, die die RecognitionService-Schnittstelle von Android umsetzt, sowie das *ExecuteUserRequestUseCase*-Interface des domain-Moduls implementiert, um durch Starten und Stoppen gesteuert werden zu können.
	ApiAiServiceHandler.kt Im Dialogflow-Service wird der User-Input direkt an das SDK übergeben und aufgerufen. `override fun getApiAiResponse(input: ↪ String): Observable<AIResponse?> { val aiRequest = AIRequest() aiRequest.setQuery(input) return Observable.defer({ ↪ Observable.fromCallable { ↪ aiDataService.request(aiRequest) } }) .retry({ _, error -> error is ↪ NullPointerException }) } override fun onResult(result: AIResponse?) ↪ { response = result }`
app	*IDEActivity.kt* - Der Unterschied der Events auf dem Button ist zu implementieren.

Modul	Anpassung

	``
	fab.setOnTouchListener { _, event ->
	 if(event.action == ACTION_DOWN){
	 idePresenter.performUserInputStart()
	 }else if(event.action == ACTION_UP){
	 idePresenter.performUserInputStop()
	 }
	 true
	}
	``

	IDEContract.kt, IDEPresenter.kt - Die Differenzierung zwischen Button drücken und Button loslassen muss der Presenter kennen.
	
	``
	interface Presenter : BasePresenter<View>
	 ↪ {
	 fun performUserInputStart()
	 fun performUserInputStop()
	``

Modul	Anpassung

```
override fun performUserInputStart() {
 userStartSpeakingUseCase.execute(
  Action {},
  Consumer<Throwable> {
   view?.error(it.message!!)
  }
 )
}
override fun performUserInputStop() {
 userStopSpeakingUseCase.execute(
  Consumer<String> {
   executeUserRequestUseCase
    ↪ .execute(WorkspaceObserver(),
    ExecuteUserRequestUseCase.Params
     ↪ .forMiniWorld(
     miniWorldProfileVMMapper
      ↪ .transformVMtoM(view?
      ↪ .getProfileModel()),it))
  },
  Consumer<Throwable> {
   view?.error(it.message!!)
  }
 )
}
```

C.11. Anpassungen Fallstudie 2 Übersicht

Modul	Anpassung
domain	*NLPResponse.kt* - Ein allgemeines *NLPResponse* Model Objekt sollte an Stelle des *AIResponse* Objektes verwendet werden.

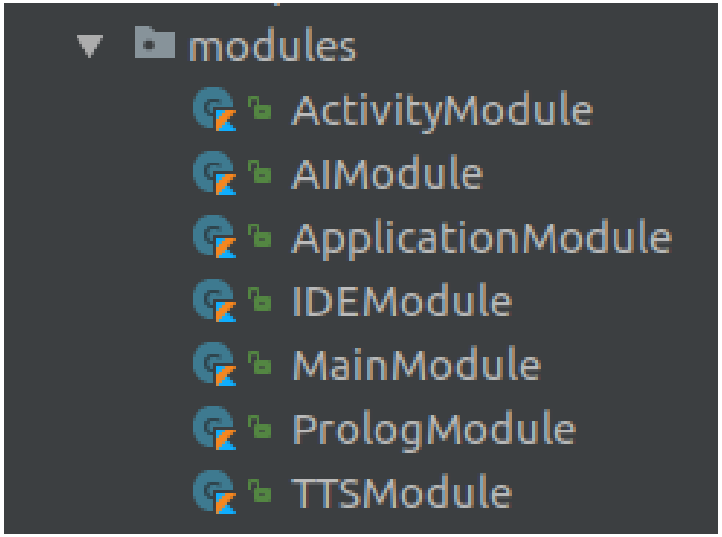
	APIAIHandler.kt - Das Interface ist namentlich am Dialogflow angelehnt. Der APIAIHandler wird in *NLPHandler* umbenannt.
	`interface APIAIHandler{`
	`fun getApiAiResponse(result:`
	`↪ AIResponse?): Observable<AIResponse?>`
	`}`

	GetAPIAILogicItemsUseCase.kt - Dieser UseCase wertet das Antwortobjekt aus und mappt dessen Ergebnisse in LogicItems. Hier müsste das Mapping auf das entsprechende Backend angepasst werden.
data	*ApiAiServiceHandler.kt* - Dieser Service-Handler implementiert das spezifische Verhalten der neuen Schnittstelle und gibt das NLPResponse Objekt zurück.

C.12. Anpassungen Fallstudie 3 Übersicht

Modul	Anpassung
app	*CommonRuleView.kt, FactView.kt, ImageRopeView.kt, KnowledgeView.kt, PropertyFactView.kt, SingleFactView.kt* - Alle ModelView-Objekte, die für die Darstellung zuständig sind, müssen angepasst, ersetzt oder erweitert werden, um die neue spezifische Darstellung zu erreichen.
	KnowledgeViewMapper.kt - Dieser Mapper ist für das Übersetzen der domain-Modul zu den data-Modul Objekten und umgekehrt zuständig (Model in ViewModel sowie ViewModel in Model). Bei veränderten ViewModel Objekten muss der Mapper diese korrekt erstellen.

C.13. Auswertungsübersicht Fallstudien

	Kategorie	Ausprägung	Beschreibung	Beobachtung
3a	Software Produktqualität Wartbarkeit - Modularity	Stark modular	Viele kleine Module die einzelne Aufgaben übernehmen	Ein Abhängigkeitsgraph wurde mit Hilfe des Frameworks Dagger2 (https://google.github.io/dagger/) erstellt. So können zyklische Abhängigkeiten und Boilerplate-Quellcode verhindert werden.  **Fallstudie 3:** Beim Anpassen der View muss nur das Modul *app* angepasst werden.
		Teilweise modular	Es existieren einzelne Module mit separaten Aufgaben	Es existieren auf oberste Ebene drei Module (app, domain, data) (siehe Abschnitt 5.4), die separate Aufgaben übernehmen.

	Kategorie	Ausprägung	Beschreibung	Beobachtung
				Fallstudie 2: Das Austauschen eines Backendsystems ist dadurch gut möglich, da im äußersten Ring der Clean Architecture, die Schnittstelle implementiert wird. Diese hat kaum eigene Abhängigkeiten.
		Nicht modular	Es existieren keine Module	
3b	Software Produktqualität Wartbarkeit - Modifiability	Gut veränderbar	Änderungen in der App sind ohne großen Aufwand möglich. Refactoring ist kaum bis gar nicht nötig	**Fallstudie 1:** Änderungen müssen in jedem der Module durchgeführt werden, sind aber hauptsächlich hinzuzufügen Funktionen. **Fallstudie 2:** Die Schnittstelle zum Backend wird in der entsprechenden ServiceHandler-Klasse ausgetauscht. **Fallstudie 3:** Es müssen die Objekte verändert werden, die direkt mit der Anforderung zusammenhängen. Kein weiteres Refactoring nötig.
		Veränderbar	Änderungen in der App sind möglich. Refactoring ist nicht zum Teil nötig	**Fallstudie 2:** Durch die Verwendung des Dialogflow Response-Objektes, muss im domain-Modul Refactoring verwendet werden.

	Kategorie	Ausprägung	Beschreibung	Beobachtung
		Nicht modular	Änderungen in der App sind kaum möglich. Starkes Refactoring ist nötig	
3b	Software Produktqualität Wartbarkeit - Testability	Gut testbar	Die App ist vollumfänglich gut automatisierbar testbar	
		Testbar	Die App ist mit überschaubarem Aufwand automatisierbar testbar	Abschnitt 5.4.5 beschreibt, wie Unit-Testing für das domain-Modul (Businesslogik) durchgeführt wird.
		Nicht testbar	Automatisiertes Testen erzeugt einen so hohen Aufwand, dass es keine sinnvolle Umsetzung gibt	Android liegt in verschiedensten Version, Bildschirmgrößen, OEM-Änderungen vor. Äquivalenzklassen zu finden ist schwierig. View-Funktionalitäten nehmen große Anteile im Vergleich zur Businesslogik ein.

D. Datenträger

Der angefügte Datenträger enthält zum einen Material, das für den Anhang zu groß ist und zum anderen entstandene Audioaufnahmen und Softwareartefakte. Eine PDF-Version dieser Arbeit ist dort ebenfalls vorhanden. Die Pfad-Struktur ist möglichst einfach gehalten und geht lediglich eine Ebene tiefer. Die Ordner sind *Fragebogen*, *Quellcode* und *Aufnahmen*. In der PDF-Version auf dem Datenträger sind die Pfade zu den digitalen Anhängen direkt verlinkt und können durch diese aufgerufen werden.

D.1. PDF-Version der Masterarbeit

Die PDF-Version der Masterarbeit befindet sich im root-Verzeichnis.
Absoluter Pfad:
/
Dateien:
Masterthesis_Marcel_Kaufmann.pdf

D.2. Fragebogen

Im Unterordner *Fragebogen* befinden sich die von den SuS ausgefüllten Fragebogen.
Absoluter Pfad:
/Fragebogen
Dateien:
/Fragebogen/Fragebogen1.pdf
/Fragebogen/Fragebogen2.pdf
/Fragebogen/Fragebogen3.pdf
/Fragebogen/Fragebogen4.pdf
/Fragebogen/Fragebogen5.pdf
/Fragebogen/Fragebogen6.pdf
/Fragebogen/Fragebogen7.pdf
/Fragebogen/Fragebogen8.pdf
/Fragebogen/Fragebogen9.pdf
/Fragebogen/Fragebogen10.pdf

D.3. Quellcode

Im Unterordner *Quellcode* befinden sich die software-relevanten Artefakte. Zum einen der Export des Dialogflow-Assistenten und eine Historie der geloggten Anfragen und zum anderen die App als Kompilierte apk-Datei sowie dessen Quellcode.

Absoluter Pfad:

/Quellcode

Dateien:

/Quellcode/Dialogflow_Export_LogicIDE_1.zip
/Quellcode/dialogflowHistory.txt
/Quellcode/Masterthesis_LogicIDE-master-1.0.71.zip
/Quellcode/SLIDE_release-1.0_master-1.0.71_06-12-2017.apk

D.4. Aufnahmen

Im Unterordner *Aufnahmen* befinden sich die drei erstellten Audioaufnahmen. Dies ist das Experteninterview sowie die beiden Aufnahmen der Gruppenarbeit zweier Arbeitsgruppen.

Absoluter Pfad:

/Aufnahmen

Dateien:

/Aufnahmen/AufnahmeGruppenarbeit1.mp3
/Aufnahmen/AufnahmeGruppenarbeit2.mp3
/Aufnahmen/ExpertenInterview1.mp3

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere, dass ich alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe, und dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist.

_____ _____

Ort, Datum Unterschrift